



**AN OBJECT-ORIENTED APPROACH TO
THE MODELING AND VISUALIZATION OF
EARLY-STAGE BREAST CANCER TUMORS**

THESIS

Bruce C Jenkins, Captain, USAF

AFIT/GOA/ENS/00M-04

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC QUALITY INSPECTED 4

20000613 096

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE AN OBJECT-ORIENTED APPROACH TO THE MODELING AND VISUALIZATION OF EARLY-STAGE BREAST CANCER TUMORS			5. FUNDING NUMBERS	
6. AUTHOR(S) Bruce C Jenkins, Captain, USAF				
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GOA/ENS/00M-04	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL / HEPA Attn: D. Wilson 2800 Q Street, Building 824 WPAFB OH 45433 DSN: 785-3122			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES Dr. Kenneth W Bauer, Jr., ENS, DSN: 785-3636, ext. 1234 / Kenneth.Bauer@afit.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED			12b. DISTRIBUTION CODE	
ABSTRACT (Maximum 200 Words) <p>Great strides have been made in controlling the progression of breast cancer, but medical professionals continue to rely primarily upon traditional mammography for early detection. Enhancing breast cancer detection capabilities—and thus reducing tumor detection time—should result in a reduction in mortality rates. The detection capabilities of mammography may be enhanced through improved modeling and visualization techniques.</p> <p>This thesis is an extension of initial research conducted at the Air Force Institute of Technology. Previous efforts focused on developing a mathematical model for simulating the growth of cancer within unconstrained 3-dimensional space. The research also explored the behavior of the model as it interacts with simulated breast tissue structures.</p> <p>This effort implements improvements suggested by the previous research and explores alternative modeling approaches. These approaches are implemented using object-oriented software engineering techniques within the Java programming environment. The model is first replicated in unconstrained 2- and 3-dimensional space, and then embellished to more closely model the semi-autonomous behavior of cancerous cells. We determine that Java provides a suitable environment for simulating tumor growth, and this study concludes with a model built upon a rule set for controlling cell behavior.</p>				
14. SUBJECT TERMS Cancer, Breast Cancer, Tumor, Modeling, Object Oriented, Java, Simulation, Visualization			15. NUMBER OF PAGES 147	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U. S. Government.

AFIT/GOA/ENS/00M-04

AN OBJECT-ORIENTED APPROACH TO THE MODELING AND
VISUALIZATION OF EARLY-STAGE BREAST CANCER TUMORS

THESIS

Presented to the Faculty

Department of Operations Research

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Operational Analysis

Bruce C Jenkins, B.S.

Captain, USAF

March 2000

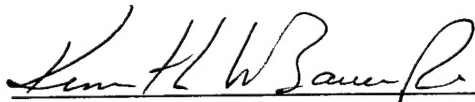
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFTT/GOA/ENS/00M-04

AN OBJECT-ORIENTED APPROACH TO THE MODELING AND
VISUALIZATION OF EARLY-STAGE BREAST CANCER TUMORS

Bruce C Jenkins, B.S.
Captain, USAF

Approved:


Kenneth W. Bauer, Jr., Professor (Chairman)

6 MAR 00
date


J.O. Miller, Lt Col, USAF

6 Mar 00
date

Acknowledgements

I would like to thank Dr Kenneth Bauer for seeming to know when and how to push my buttons in order to keep me moving not simply *forward*, but in the proper direction as well. This thesis was yet another valuable AFIT learning experience; the lessons learned will benefit me for years to come. I would also like to thank Lt Col J.O. Miller for his humor and encouragement. Our weekly thesis meetings would not have been the same without his presence. Thanks also to Capt C. Brian Bassham, whose previous work led me on this path. His continuing research in the area of modeling and visualization proved an effective motivator for the task that lay before me.

Finally, I would like to thank those who made the greatest sacrifice while I pursued yet another goal: my dedicated wife, Susan, and our loving daughter, Kayla. To these two I am forever indebted. Their support is unwavering; their patience, unending... I love you both.

Bruce Jenkins

List of Figures

Figure 3-1: The incremental model (Source: Pressman, 1997).....	23
Figure 4-1: Tumor01 Class Diagram	29
Figure 4-2: Cancer cell movement within the 2-D tumor growth space	32
Figure 4-3: Cell vector before first population doubling	34
Figure 4-4: Cell vector after first population doubling (step 4)	35
Figure 4-5: Cell vector after second population doubling (step 7)	36
Figure 4-6: Tumor01 Object relationship diagram	37
Figure 4-7: Tumor02 Class Diagram	39
Figure 4-8: Tumor02 Object Relationship Diagram	42
Figure 4-9: Tumor03 Class Diagram	46
Figure 4-10: Tumor03 Object Relationship Diagram	53
Figure 4-11: Tumor03_1 modifications to CancerSimApplet class.....	56
Figure 4-12: Tumor04 Class Diagram	57
Figure 4-13: Tumor04_2 Class Diagram	59
Figure 4-14: CancerSim Control Panel (Tumor04, Tumor04_1).....	64
Figure 4-15: Tumor04_1 Object Relationship Diagram	65
Figure 4-16: Simulation flow chart (Tumor04_2).....	67
Figure 5-1: Tumor01 population doubling (expected)	70
Figure 5-2: Tumor01 population doubling (actual).....	70
Figure 5-3: Tumor01 results.....	71
Figure 5-4: Tumor02 results [left] compared to Bassham's Tumor5	72
Figure 5-5: Tumor03 results.....	73
Figure 5-6: Tumor03_1 density projections along the x-, y-, and z-axes	75
Figure 5-7: Tumor04 GUI and DOS window	76
Figure 5-8: Tumor04 sample DOS box output	77
Figure 5-9: Tumor 04 [upper projections] and Bassham's Tumor3d [lower projections].	78
Figure 5-10: Tumor04 simulation results.....	81
Figure 5-11: Tumor04_1 simulation results.....	81
Figure 5-12: Comparison of simulation run times	83
Figure 5-13: Simulation results (Tumor04_2)	85

List of Tables

Table 2-1: Government Web resources.....	9
Table 2-2: University Web resources.....	10
Table 2-3: Breast Cancer Staging (Source: <i>The Breast Cancer Digest</i>)	11
Table 3-1: Bassham's tumor growth algorithm	23
Table 4-1: Attribute List (Tumor01.TumorApplet1)	30
Table 4-2: Attribute List (Tumor02.CancerSimApplet)	40
Table 4-3: Attribute List (Tumor03.CancerSimApplet)	49
Table 4-4: Lookup table used in class CellDirection	52
Table 4-5: CancerSim model summary.....	68
Table 5-1: Mean population-doubling times (in seconds)	82

Listings

Listing 4-1: Determining cell direction in 2-D space.....	33
Listing 4-2: Thread Creation in Tumor03	51
Listing 4-3: Tumor Growth Algorithm in Tumor03	52
Listing 4-4: Determining the maximum contrast for a tumor projection.....	61
Listing 4-5: Tumor growth process for 3-D embellishments	63

Table of Contents

ACKNOWLEDGEMENTS.....	iii
LIST OF FIGURES	iv
LIST OF TABLES	v
LISTINGS.....	vi
TABLE OF CONTENTS.....	vii
ABSTRACT	ix
I. INTRODUCTION	1
1.1 BACKGROUND.....	1
1.2 STATEMENT OF THE PROBLEM	2
1.3 RESEARCH FOCUS	3
1.4 CONTRIBUTION OF RESEARCH.....	4
1.5 ASSUMPTIONS, RISKS, AND LIMITATIONS	4
II. LITERATURE REVIEW.....	6
2.1 INTRODUCTION	6
2.2 RELEVANT INTERNET RESOURCES	8
2.2.1 <i>Government</i>	8
2.2.2 <i>Education</i>	10
2.2.3 <i>Other</i>	11
2.3 STAGES OF BREAST CANCER.....	11
2.4 SCREENING FOR EARLY DETECTION	12
2.4.1 <i>Physical Examination</i>	12
2.4.2 <i>Mammography</i>	13
2.4.3 <i>Ultrasound</i>	14
2.4.4 <i>Magnetic Resonant Imaging (MRI)</i>	14
2.4.5 <i>Other Imaging Techniques</i>	16
2.5 BREAST CANCER MODELING	16
2.5.1 <i>Complex Adaptive Systems</i>	16
2.5.2 <i>Molecular</i>	18
2.5.3 <i>Mathematical</i>	18
2.5.4 <i>Other Models</i>	19
III. METHODOLOGY	21
3.1 OVERVIEW	21
3.2 JAVA CLASS DEVELOPMENT	24
3.3 GUI DEVELOPMENT	24
3.4 EMBELLISHMENTS.....	25

IV. MODEL DESCRIPTIONS	26
4.1 OVERVIEW	26
4.2 TWO-DIMENSIONAL TUMOR MODEL	27
4.2.1 Overview.....	27
4.2.2 <i>Class Development and Modeling Diagrams</i>	28
4.2.3 <i>Operational Description</i>	30
4.2.4 <i>Embellishments</i>	38
4.2.5 <i>Summary</i>	43
4.3 THREE-DIMENSIONAL TUMOR MODEL	44
4.3.1 Overview.....	44
4.3.2 <i>Class Development and Modeling Diagrams</i>	46
4.3.3 <i>Operational Description</i>	49
4.3.4 <i>Embellishments</i>	54
4.3.5 <i>Summary</i>	68
V. ANALYSIS AND RECOMMENDATIONS.....	69
5.1 OVERVIEW	69
5.2 TWO-DIMENSIONAL TUMOR MODELS.....	69
5.3 THREE-DIMENSIONAL TUMOR MODELS.....	72
5.4 CONCLUSION	86
5.5 RECOMMENDATIONS	87
APPENDIX A: JAVA SOURCE CODE - TUMOR04	90
APPENDIX B: JAVA SOURCE CODE - TUMOR04_2.....	106
GLOSSARY	124
BIBLIOGRAPHY	131
VITA.....	134

Abstract

Great strides have been made in controlling the progression of breast cancer, but medical professionals continue to rely primarily upon traditional mammography for early detection. Enhancing breast cancer detection capabilities—and thus reducing tumor detection time—should result in a reduction in mortality rates. The detection capabilities of mammography may be enhanced through improved modeling and visualization techniques.

This thesis is an extension of initial research conducted at the Air Force Institute of Technology. Previous efforts focused on developing a mathematical model for simulating the growth of cancer within unconstrained 3-dimensional space. The research also explored the behavior of the model as it interacts with simulated breast tissue structures.

This effort implements improvements suggested by the previous research and explores alternative modeling approaches. These approaches are implemented using object-oriented software engineering techniques within the Java programming environment. The model is first replicated in unconstrained 2- and 3-dimensional space, and then embellished to more closely model the semi-autonomous behavior of cancerous cells. This study concludes with a model built upon a rule set for controlling cell behavior.

AN OBJECT-ORIENTED APPROACH TO THE MODELING AND VISUALIZATION OF EARLY-STAGE BREAST CANCER TUMORS

I. Introduction

1.1 Background

Breast cancer continues to be the leading cause of cancer deaths among American women. This fact remains, despite a significant amount of medical, scientific, and financial resources having been applied to finding a cure for cancer. Nearly 10 years ago there was a refocusing of federal cancer research efforts by the Secretary of Health, Public Health Service (PHS). The Secretary directed the PHS to develop what is known as the National Strategic Plan for the Early Detection and Control of Breast and Cervical Cancers. The National Strategic Plan provides a framework for national health organizations to coordinate their research efforts. This framework in turn focuses attention on the specific health needs of American Women (U.S. Department of Health and Human Services [USDHHS], 1993: 1).

Generally, the plan is designed to present strategies for implementing research agencies to use as a model. Specifically, the plan addresses the following issues: Integration and Coordination, Public Education, Professional Education and Practice, Quality Assurance for Breast Cancer Screening, Quality Assurance for Cervical Cancer Screening, and Surveillance and Evaluation. Combined, the efforts to improve agency coordination and quality while enhancing national education appear to have shown positive results. In a 1998 Boston Globe article, a former National Cancer Institute (NCI) chief speculated that breast cancer would be reduced by 30 percent over 1971 levels

(Saltus, 1998). An increase in breast cancer awareness and medical screening alone has produced an increase in detection, and thus an improvement in mortality rates, since 1973 (USDHHS, 1993: 5).

Even with this apparently successful effort to reduce the mortality caused by breast cancer, relatively little research has been done on early *detection* of breast cancer. By the time a cancerous tumor is physically detected, there is increased risk that the cancer will metastasize, or spread, to other parts of the body. When tumors are too small to be physically detected, they can be detected through mammography—essentially a breast x-ray. Reading a mammogram is subject to error, however. An outright misreading of the mammogram is possible, or variations in breast tissue density may cause the tumor to be masked or otherwise hidden. If a woman's chance of surviving the breast cancer is to be increased, these detection difficulties must be overcome.

Researchers are now attempting to enhance the effectiveness of mammography through digital techniques (*Women's Health Weekly*, 1999). Digital mammography allows for significant improvements in contrast and resolution of mammograms. In addition to enhancements provided by digital mammography, computer software systems have been developed to "read" an image and provide recommendations to the radiologist (*Cancer Weekly*, 1999; *Women's Health Weekly*, 1999). Further, additional research into magnetic resonance imaging (MRI) is being done with the expectation that the MRI can provide a clearer, more accurate image than mammography (*Chain Drug Review*, 1999).

1.2 Statement of the Problem

While great strides have been made in controlling cancer's ability to progress, the medical community continues to rely primarily upon traditional mammography or

magnetic resonant imaging (MRI) techniques for early discovery of tumors. Enhancing breast cancer detection capabilities—and thus reducing tumor detection time—should result in a reduction in mortality rates. The detection capabilities of mammography may be enhanced through improved modeling and visualization techniques.

1.3 Research Focus

This problem is an extension of the research conducted by Captain C. Brian Bassham (GOA-99M), whose efforts focused on developing a solid mathematical model of a cancerous growth within the female breast. Bassham identified some weaknesses in the model, suggested several improvements, and provided examples of areas where the modeling environment itself may be enhanced as well.

The scope of this effort, therefore, will be to build upon the model developed by Bassham. This will be accomplished using the Java platform from Sun Microsystems. There are several reasons for translating Bassham's model to Java: First, complete portability of the tumor model is a very desirable property. Model portability allows for further development beyond this research effort on virtually all computer platforms available today, from personal computers to mainframes. The second reason for using the Java environment is that it is a pure object-oriented (OO) language and has the power of other OO languages, such as C++, but without the memory management pitfalls that can hamper the debugging process. Additionally, Java3D, a sophisticated Java extension for 3-dimensional visualization, appears to be as robust as special purpose environments, such as the Visual Toolkit (VTK). While offering the same robustness, it exceeds the VTK's known capabilities in that Java3D may be incorporated directly in the tumor model. This provides not only the portability already mentioned, but dispenses with the

requirement to manage intermediate data that is an artifact of the MatLab™/VTK model. As a final, albeit minor, consideration, Java is not only omnipresent, it is absolutely free. This allows anyone to contribute to this line of research with no financial investment. In addition to the Java development kit being widely available at no cost, there are several Java integrated development environments freely available that can streamline and expedite the development process.

1.4 Contribution of Research

The long-term goal of this research effort is to assist medical professionals in detecting breast cancer much earlier in the tumor growth process. The short-term results of this effort should allow follow-on researchers to simulate the growth of a single tumor on any computer platform supporting the Java Runtime Environment (JRE).

1.5 Assumptions, Risks, and Limitations

Since the precise cause of breast cancer is not yet known, and the behavior of breast cancer tumors not fully understood, that we have a “reasonable” model representation of breast cancer will be assumed. The risk with this assumption is that the cancer cell object model may under- or over-characterizes a real-world biological cell. This may result in a model that is too simple and lacks the characteristics necessary to produce data in which we might have a reasonable level of confidence. On the other hand, the model may be so complex that it simply does not work—it either produces no data or the data cannot be interpreted.

The standard Java math libraries (classes) are assumed to be sufficiently robust and will be adequate for the mathematical modeling of a cancer tumor. For the purpose of simply translating the existing MatLab™ code to Java, there appears to be little risk.

The scope of this thesis effort should prevent encountering any purely mathematical limitations with the Java math libraries.

It must be assumed that the skills required to implement a complex mathematical model in the Java environment will be mastered in the time allowed for this project. The risk is only slight that any particular problem encountered cannot be resolved either in-house or with the assistance of outside resources. In general, none of the risks outlined above are extraordinary.

II. Literature Review

2.1 Introduction

The amount of information available concerning breast cancer research, prevention, and discovery on the Internet and in libraries is overwhelming. The Internet provides the opportunity to obtain up-to-date information quickly and on a broad category of topics. Despite this positive aspect, it is discouraging to find such a high ratio of irrelevant material to that which is relevant. Nevertheless, the Internet proved to be an invaluable first stop for this particular research effort. Not only is historical and traditionally hard copy information becoming increasingly available in electronic form via the Internet, but late-breaking research and news are now widely available in readily digestible form.

While Internet-harvested information was used extensively in the research, it is primarily relegated to a supporting role in this thesis. Some hard copy material was obtained from the Air Force Institute of Technology's Academic Library, but a majority of the books and journals were found at Wright State University's Fordham Medical Sciences and Dunbar libraries.

If there is no desire or inclination to conduct research along the path presented here and by Bassham, but one wishes to gain an understanding of breast cancer and scope of its impact, then two breast cancer literature resources stand out. The first reference, *The Breast Cancer Digest*, contains detailed information on breast cancer, how it develops, how it is diagnosed and treated, and lists breast cancer support services to aid those who may be impacted by the disease. Despite the depth of its coverage, it is mostly devoid of medical and technical terminology not familiar to the layperson. And although

the guide itself is dated, this simply impacts the relevance of the statistics and nullifies comments concerning recent developments in diagnosis. The second outstanding reference is the *Atlas of Breast Cancer*. While this guide contains information more technically challenging than the previous reference, it also contains a vivid pictorial of breast cancer's victims and the results of treatment used to contain its advancement. Combined, these two references summarize every aspect of breast cancer and ultimately paint an explicit picture illustrating the urgency in discovering a means of preventing this disease.

This chapter will first describe some of the Internet resources found to be timely and relevant to this thesis. It is important to note that these particular sources most likely can be relied up for future research or simply as points of interest. Additionally, while tens of sites proved to be useful, only those Internet resources found to be the *most* useful are included in this discussion.

After a description of the Internet resources, the causes, or contributing factors, to breast cancer will be described. Next, some general life-cycle depictions of breast cancer will be presented. We will subsequently review the current efforts in detecting breast cancer at the earliest possible stages of development. Finally, some models of the behavior of breast cancer will be described.

The reader will note that a glossary exists at the back of this thesis. While not all glossary entries are used in this thesis, the glossary is representative of words and terminology relevant to the medical aspect of the supporting research.

2.2 Relevant Internet Resources

2.2.1 Government

Some of the most interesting and information-rich resources were found at Web sites maintained by the United States federal government. The National Institute of Health (NIH) is intended to serve in the interest of better health for all Americans. It strives to accomplish this by conducting its own research and supporting the research of other, non-governmental organizations. The NIH can be thought of as a national clearinghouse for health issues, and it proved to be an excellent jumping off point for health information resources in general and cancer related searches in particular, by way of the National Cancer Institute (NCI). The Health Information link provides access to numerous resources directly related to this research: full-text consumer health publications, women's health issues, clinical trials, health literature references, as well as "MEDLINEplus," which is claimed to be one of the largest on-line medical libraries.

The NCI maintains resources on virtually all known forms of cancer. It is particularly useful for obtaining up-to-date press releases, which offer immediate notification of late-breaking news that might impact the direction of research efforts related to breast cancer. Additionally, a host of research initiatives and funding opportunities can be obtained from this site. From the NCI's site, the next logical source to investigate is CancerNet™. CancerNet™ is billed as a source of information for "Health Professionals, Patients and the Public." From this location, one can obtain cancer information applicable to the layperson, the health professional, or researchers. Of particular interest here is the CANCERLIT® bibliographic database. The search capabilities provided by CANCERLIT® provide an immediate jumpstart for breast cancer

research. There is an option for limiting the search to only those records containing an abstract. Searching on “breast cancer,” for example, returned over 10,000 abstracts with bibliographic entries from 1996 to present. A search on “breast cancer modeling” reduced the number of potential references to just over 2,000. “Breast cancer detection” returned nearly 3,600 references, while “breast cancer screening” provided just over 1,900 candidate references. “Mammography,” “breast cancer,” and “microcalcification” also proved to be significant keywords within this extremely useful database.

Additional government Internet resources fall in the military domain. A discovery made early in this research was at the Department of Defense Congressionally Directed Medical Research Programs (CDMRP) site. The CDMRP is a consortium of military, scientists, physicians, and the community, and serves as a point of convergence for medical research. The CDMRP has numerous research programs, including breast cancer. Only funding offered the National Cancer Institute exceeds the level of funding for the CDMRP’s breast cancer research program (BCRP). The BCRP offers a unique perspective on breast cancer research within the Department of Defense (DoD). As program executive, the BCRP may allocate monies for breast cancer research, and the potential exists for funding of military research efforts. The government agencies covered above are summarized in Table 2-1.

Table 2-1: Government Web resources

Government Agency	Universal Resource Locator
CANCERLIT [®]	http://cnetdb.nci.nih.gov/cancerlit.shtml
CancerNet [™]	http://cancernet.nci.nih.gov/
Department of Defense Congressionally Directed Medical Research Programs (CDMRP)	http://cdmrp.army.mil/
National Cancer Institute (NCI)	http://www.nci.nih.gov/
National Institute of Health (NIH)	http://www.nih.gov/health/

2.2.2 Education

Searching for literature on cancers in the medical domain returns a long list of potential sites. Much insight into leading-edge cancer research can be garnered through a review of the many medical research facilities in the United States. The Vanderbilt University Medical Center provides a state-by-state listing of on-line medical schools. Each of the 125 links in the list was visited if possible. That is, if an error did not occur indicating the link was incorrect or the server was not available. Sites with the following characteristics are listed Table 2-2: (1) Breast cancer was featured prominently in the main page, (2) a “research” link from the main page led to breast cancer information, or (3) relevant information was found with a simple “breast cancer” keyword search. The table is a summary list of medical universities with Web sites where breast cancer research information could be found reasonably quickly.

Table 2-2: University Web resources

University	Universal Resource Locator
Huntsman Cancer Institute at the University of Utah	http://www.hci.utah.edu/
Johns Hopkins Breast Center (John Hopkins University)	http://www.med.jhu.edu/breastcenter/
Loma Linda University Cancer Institute (LLUCI)	http://www.llu.edu/llu/ci/professional/info.html
Olson Center for Women's Health of the University of Nebraska Medical Center	http://www.unmc.edu/Olson/brstdir.htm
Rita J. & Stanley H. Kaplan Comprehensive Cancer Center, New York University (NYU) School of Medicine	http://health-www.med.nyu.edu/BreastCare/
The University of Medicine and Dentistry of New Jersey (UMDNJ) and Coriell Research Library	http://www4.umdj.edu/camlbweb/brestcan.html
University of Arkansas for Medical Sciences	http://www.uams.edu/cop/rxbreastca/default.htm
University of Pennsylvania Cancer Center (OncoLink)	http://cancer.med.upenn.edu/
Vanderbilt University Medical Center	http://www.mc.vanderbilt.edu/~aubrey/medstu/medical_schools.html http://www.mc.Vanderbilt.Edu/vumc/centers/cancer/html/programs_breast.html

2.2.3 Other

Internet sites not falling in government, military, or educational domains include for-profit commercial and other organizations. There were many useful sites; too many to be listed here, in fact. Many of the Web resources not falling in the category of government, military or educational institutes can be found through the sites listed in Table 2-1 and Table 2-2. One site that cannot be overlooked, however, is the obvious: the America Cancer Society. See http://www3.cancer.org/cancerinfo/load_cont.asp?ct=5 for information specifically related to breast cancer.

2.3 Stages of Breast Cancer

The literature is consistent in describing the stages of breast cancer as it develops and spreads. "Staging" generally refers to the size of a tumor and the extent to which it has metastasized, or spread to other parts of the body. Metastasis can be aggregated into three regions: local, regional, and distant. Local is confined to the initial site. Regional metastasis describes breast cancer that has spread to the lymph nodes. Distant metastasis is breast cancer that spread beyond the lymph nodes. Table 2-3 summarizes the stages of breast cancer, which was published in 1982 by the American Joint Committee TNM (tumor, lymph node, and distant metastases) Staging of Breast Cancer.

Table 2-3: Breast Cancer Staging (Source: *The Breast Cancer Digest*)

Stage	Tumor Size (cm)	Metastases
I	< 2	local; regional negative; distant <i>not</i> detected
II	2-5	local; regional negative; distant <i>not</i> detected; <i>or</i>
	< 5	local and regional; distant <i>not</i> detected
III	> 5	<i>Or</i>
	any	invasion of skin or chest wall or "grave signs" ¹ , <i>or</i>
	any	regional (collarbone area); distant <i>not</i> detected
IV	any	local; regional positive or negative; distant detected

¹ Advance disease indicators such edema, skin ulceration and pitting, or satellite skin nodules (National Cancer Institute; 1984: 47).

2.4 Screening for Early Detection

From the stages identified in the previous section, it is apparent that improving a woman's chance of surviving breast cancer relies heavily upon detecting the cancer at the earliest possible point of tumor development. Attempts are being made to predict a woman's risk of contracting breast cancer by focusing on enhancing mathematical models that identify this risk (Rosner, 1996). The values of the model's parameters are based upon a woman's personal and genetic history. If the model indicates a woman is "at risk," then she can be monitored more closely so that if cancer does develop, the probability of detecting the cancer at an early stage is improved. If a woman is determined to have an increased risk of developing breast cancer, she is given recommendations regarding the frequency of conducting a breast self-examination (BSE) and clinical breast examination (CBE), to include "visual" exams, such as mammography, ultrasound, and magnetic resonant imaging. This falls in the area of screening, and the literature is rich with information on both the rationale and the methods for screening for the early detection of breast cancer.

2.4.1 Physical Examination

The U.S. Department of Health and Human Services has stated in numerous publications that women should make breast examinations a regular part of their medical checkup routine. Barton *et al* reported in *The Journal of the American Medical Association* that the CBE is a critical component of early detection (JAMA, 1999). Their study reviewed MEDLINE data on breast examinations from 1966-1997 and is one of the

most thorough reports on the effectiveness of the CBE found in the literature. Not only does their report analyze the effectiveness, but also discusses the basis for the physical breast exam, the risk factors involved, various examination methods, and their accuracy. The CBE appears to be most effective at detecting breast cancer when combined with mammography.

2.4.2 Mammography

Traditional mammography involves compressing the female breast in an apparatus and subsequently subjecting the breast to a small amount of X-ray radiation (less than 1 rad). After the radiation passes through the breast, it strikes and exposes X-ray film. The resulting image is analogous to a black-and-white photographic negative. Both vertical and horizontal compression images are made. Nonpalpable breast anomalies may be detected through traditional mammography, and, as stated in the previous paragraph, is designed to compliment a physical breast examination. Sequences of mammograms taken over time can indicate subtle breast tissue differences not readily attributable to changes expected as a result of aging.

To improve the overall effectiveness of mammography, various techniques are employed to enhance the resulting image. One technique involves taking existing mammographic film and placing it in a reader that scans and digitizes the image. The benefit of this technique is that computer software can provide contrast enhancement and detection algorithms to point out suspicious areas to physicians (*Cancer Weekly Plus*, 08 February 1999; *Women's Health Weekly*, 14 June 1999). A second article in *Women's Health Weekly* discusses an alternative to scanning traditional film. True "digital

mammography,” where the X-ray exposure results directly in a digital image, is being welcomed as a potential replacement to film mammography.

2.4.3 Ultrasound

The use of “echoes” to detect objects, most notably through sound navigation and ranging, or SONAR, is not limited to submarines or under-sea mammals. Ultrasound is routinely used to monitor a woman’s fetus or provide therapy to deep tissue injuries. For breast imaging, high frequency sound waves are transmitted into the breast and their echoes recorded and converted to computer images. The use of ultrasound is limited when compared to mammography, but *The Breast Cancer Digest* indicates that it is particularly useful for identifying cysts, and is frequently used for examining younger women, whose breast structure is typically denser than the breast tissue of older women (1984: 39). The success of ultrasound in distinguishing cysts from solid masses is support by a more recent study by Saarela, *et al* (1998: 1). Both *The Breast Cancer Digest* and the Saarela article express that ultrasound is generally not used as a stand-alone technique; rather, it is an alternative technique used to supplement mammography. While the use of ultrasound to supplement mammography is widespread, it is not the only supplemental technique available.

2.4.4 Magnetic Resonant Imaging (MRI)

The use of MRI in detecting human internal abnormalities is not new. The use of MRI specifically for detecting breast cancers is relatively new, however. As research and the results of that research progress, more can be found in the literature regarding the impact of MRI on the effectiveness of breast cancer detection.

A majority of the recent literature on the use of MRI in detecting breast cancer was found on-line through OhioLink™, via the WSU library on-line catalog system (<http://www.libraries.wright.edu/>). An electronic article from the 11 May 1999 issue of *Medical Industry Today*, retrieved through LEXUS-NEXUS, cited that, in one particular study, an 89 percent rate for detection and stage determination of lobular breast cancer. Lobular breast cancer is one of the more difficult variations of female breast cancer to detect using traditional mammography or ultrasound techniques. The assertion of the value of MRI in uncovering tumors hidden in the “normal tissue” of a mammogram is also cited in *Women’s Health Weekly* (“MRI vs. Mammography and Ultrasound for Cancer,” 10 May 1999).

The high interest in MRI as a means of detecting breast cancers missed by mammography is further supported in a 29 July 1999 excerpt from *Radiology* (“MRI Detects Breast Tumors Missed by Mammography”). The cost of MRI exceeds that of mammography by a factor of 10 and has difficulty identifying tumors at 2 mm and smaller. Nevertheless, it is considered a viable alternative to mammography, particularly when high sensitivity is desired. Work by Davis and McCarty summarizes a tremendous literature review of the sensitivity and effectiveness of MRI (1997). The data from this study support assertions that MRI is extremely useful when improved accuracy of breast cancer staging is critical to determining a prognosis and treatment path. Despite the superior imaging capabilities of MRI, it is still considered a supplement to mammography and ultrasonography (1997: S296).

2.4.5 Other Imaging Techniques

Confidence in established imaging techniques has not impeded the development of newer methods for increasing the probability of correctly diagnosing breast cancer. A paper release by the Food and Drug Administration describes a hand-held device called the "T-Scan" that uses electrical current to measure variations in the density of targeted areas of the breast ("FDA Approves New Breast Imaging Device," 19 Apr 1999). The device is not designed to replace conventional mammography, rather, to augment mammography when some ambiguity exists during diagnosis. A paper by Simonetti, *et al*, also discusses alternative imaging techniques, including digital mammography, digital tomosynthesis, digital subtraction angiography (DSA), and computed tomography laser mammography (CTLM). The Simonetti paper concludes, however, that mammography remains the imaging technique of choice due to its sensitivity and cost effectiveness (1998: S241).

2.5 Breast Cancer Modeling

2.5.1 Complex Adaptive Systems

Modeling cancer as a complex adaptive system (CAS) is an intriguing proposition and, as yet, is unexplored. A cancerous tumor can be thought of as a tightly knit group of "out of control" cells; they do not respond to the body's regular "divide" and "stop dividing" signals. That is, these renegade cells behave in a manner inconsistent with good cell "order and discipline." Establishing their own source of nutrition, they appear to move, divide, and spread by their own will, autonomous of the body's control mechanisms.

Pitot, in his explanation of carcinogenesis, describes a neoplasm as an autonomous entity, somewhat independent of the rules applied to neighboring, normal cells (1978: 16). While his writing is meant for those interested in the language of oncology, his message can be considered in the context of a CAS. The apparent autonomous behavior of a tumor captures the essence of a CAS. There appears to be a wealth of information regarding complex adaptive systems within the operations research community. Very few references were found, however, that fit within the context of medical research in general or cancer research in particular.

Coffey's article on complexity alludes to CAS-like properties when he describes the "self-organization" potential of a tumor during the evolution of individual cells. He discusses the possibility that the very randomness of the living systems delves into a world of chaos that is actually predictable. This discussion, which is on the fringe of chaos theory, helps paint a picture of how complexity and chaos theory might be used in the study of the behavior of cancer.

Schwab and Pienta provide greater detail on the potential for the use of chaos theory in the study of cancer. They provide an overview of the behavior of cancer prior to their discussion of complexity and chaos theory. This leads quite naturally into an argument on why cancer itself fits this paradigm so aptly and therefore why cancer can and should be studied as a CAS. This is accomplished by comparing the CAS process with the known process of a tumor. While the literature on CAS and its use in modeling cancer is extremely limited, there exists tremendous potential for additional research in this area.

2.5.2 Molecular

While modeling cancer as a complex adaptive system falls at this point in the realm of the theoretical, molecular modeling is now beyond theory and holds some promise for the early detection of breast cancer. By understanding the architecture and behavior of cancer cells at the molecular level—and understanding the cells' effects on the chemistry of the human body—researchers are finding ways of detecting the presence of cancer earlier than ever before. By then chemically attacking a cancer cell's receptors, cancer may be stopped long before detection through conventional means. This research is still in its infancy, however, and therefore information in the literature is sparse.

2.5.3 Mathematical

Iyengar's (1984) anthology presents a mixed bag of mathematical modeling and computer simulation to study the behavior of biological systems. He provides an abundance of formulae and various statistical approaches to implementing computer-based simulations that mimic complex biological systems. Of particular interest are chapters 1 through 4. The first three chapters target complex biological systems in their coverage of modeling and simulation, software engineering, and statistical techniques. Chapter 4 tackles the subject of specifically modeling cancer. Here, W. Düchting, a professor of electrical engineering at the time this collection was published, applies control systems theory to model the kinetics of cancer cells.

Interestingly, Düchting describes cancer as "...cell renewal systems which have become structurally unstable in their closed-loop control circuit" (Iyengar, 1984: 56). This view is apparent in his free, yet effective use of electronic block and circuit diagrams. He further applies the processes described by these diagrams to a

100×100 grid of cells in which he simulates the activities of three cell systems. His approach strongly parallels Bassham's use of a 2-dimensional matrix to simulate a finite number of cell population doubles.

2.5.4 Other Models

Bassham's research led to the development of both 2- and 3-dimensional mathematical models of breast cancer tumor development. The basic model was developed in MatLab™, by *The Mathworks, Inc.* As the name might imply, the greatest strength of MatLab™ ("matrix laboratory") resides in its manipulation of matrices. Bassham used the visual capabilities of the software package to visually depict his matrix solution—the result of the tumor growth. He was also able to display the model's output using The Visual Toolkit (VTK), a compilation of computer programming routines optimized for visualizing data.

Bassham's model works by first creating a matrix of dimension n^3 , the "growth space," in which location x, y, z is identified as the starting position for a cancerous cell—the start of tumor growth. To improve the efficiency of the algorithm, the search is limited to a subset of the entire growth space. This subset represents the borders of the current tumor size. The subset of the growth space is then methodically searched until a cancerous cell is discovered. At each discovery of a cancerous cell, potential directions of growth are randomly searched. These candidate locations are marked and, once all candidate locations for each cancer cell are identified, the subset growth space expands and the candidate cells become full-fledged cancer cells. The process is repeated until either (a) the number of population doublings specified during simulation initiation is reached, or (b) the limits of the growth space is reached.

Once the growth is completed, the data are visually rendered either in the VTK or MatLab™. The remainder of this report focuses on replicating Bassham's model in an alternative environment, while extending both the behavioral and visualization aspects of the tumor growth. A summary of Bassham's algorithm can be found in the next chapter.

III. Methodology

3.1 Overview

The approach taken to achieve the objectives outlined earlier will be an incremental one. This should be taken literally in the software engineering sense. That is, we will develop several complete models in a step-wise fashion so that, at the conclusion of each series of steps, or increment, a limited software product—a working model—is available for demonstration and use. Each working model will be coupled with a limited-scope embellishment that serves to achieve very specific objectives. The model and embellishment are then used as inputs to an additional iteration of the process, or model development phase. This incremental approach incorporates the long-term value of a disciplined software engineering approach with the speed and short-term benefit of prototyping. Each incremental development phase consists of the following steps: analysis, design, code, and test (Pressman, 1997: 38).

The first step is to analyze the previous model developed by Bassham. The primary short-term objective is to duplicate this model in its entirety and implement the model in the Java environment. This will be accomplished by creating an algorithm that provides a high-level view of the objects and ultimately replicates the basic behavior of Bassham's model. The next step is to create a design that incorporates as much functionality as possible.

In the design phase, we take the algorithm and elaborate the model in the Unified Modeling Language (UML). The UML will form the foundation for generic model and class descriptions that can be implemented in an object-oriented computer programming language, such as C++, Java, or Ada. For this project, the UML and supporting diagrams

will be used to develop and document the model in Java. The UML will be developed in the context of Rational Rose, a 4th Generation Language CASE (computer-aided software engineering) tool. Rational Rose has the capability to automatically generate Java source code from the constructed diagrams. Java source code is the result of the next phase: coding.

In the coding phase, we will use the UML diagrams and supporting analysis documentation to write structured Java code. The tool of choice for this project is Borland's JBuilder 3, a sophisticated integrated development environment (IDE). The Java code developed here, however, is not dependent on any particular software development tool; on the contrary, any Java-compliant development tool will be able to import and compile the code resulting from this phase. The primary purpose of using JBuilder 3 is to validate the code and develop the graphical user interface (GUI) for executing and controlling the model. Execution will occur in the test phase.

The test phase is the final step in the incremental approach to software development and, thus, the development of the models resulting from this project. All testing and debugging will be accomplished using the Java IDE mentioned in the previous paragraph. Once the model appears to be functioning as designed, it will be tested using a Web browser or Java command-line environment, as appropriate.

The result of the previous four phases is a working model. With the working model, we will repeat the four phases, using some specified embellishment as input to the first phase. This process is shown graphically in Figure 3-1.

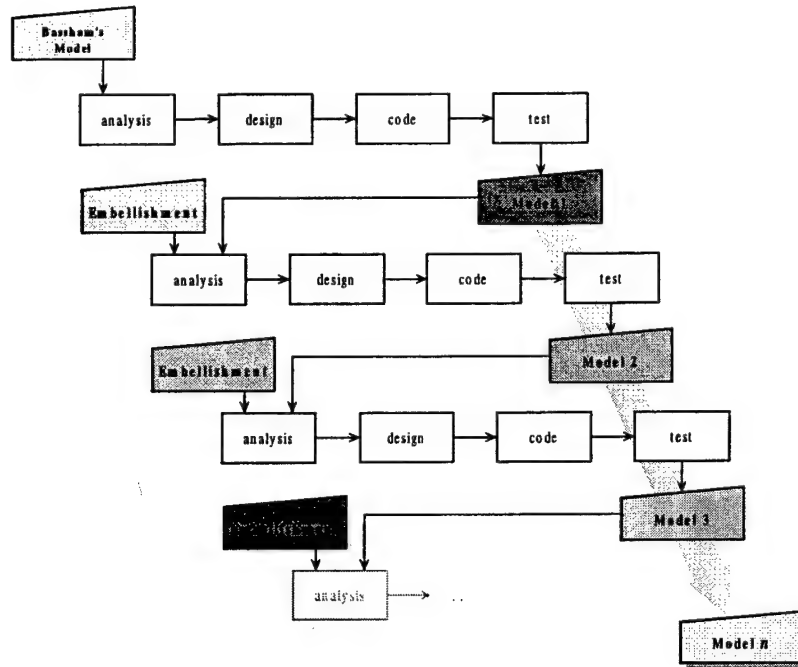


Figure 3-1: The incremental model (Source: Pressman, 1997)

We mentioned in the previous section that Bassham's basic algorithm was identified as the first step in developing a model in the Java environment. The steps outlined in Table 3-1 represent a generalization of Bassham's algorithm.

Table 3-1: Bassham's tumor growth algorithm

1. Create a 3-dimensional matrix (growth space) and initialize with 0's
2. Place the value of 2 (tumor cell) at starting position x, y, z
3. Create a virtual inner matrix (search boundaries) around the tumor cell; for the first tumor cell, the boundaries are at positions $x \pm 1$, $y \pm 1$, $z \pm 1$
4. While the number of population doubles is greater than 0, and the bounds of the virtual matrix have not exceeded the bounds of the growth space, do steps a-e; otherwise go to step 5
 - A. While the search of virtual inner matrix is not exhausted, do steps (1)-(2)
 - (1) If the matrix position being inspected contains a cancer cell, then do steps (a)-(b); else go to step (2)
 - (a) Pick a direction at random and identify a candidate cell position
 - (b) If the candidate cell position is occupied or is a barrier, go to step 4.a.(1)(a); else place the value of 1 (future cancer cell) and
 - (2) Go to step 4.A
 - B. Set all values of 1 to 2 (transform daughter cells to actual cancer cell)
 - C. Expand the bounds of the virtual inner matrix
 - D. Subtract 1 from the number of population doubles
 - E. Go to step 4
5. Display the matrix

3.2 Java Class Development

As an object-oriented (OO) language, Java is capable of modeling real-world entities within the limits of its lexicon. Classes are the foundation of these modeling capabilities. Like other OO languages, Java uses classes to characterize the encapsulation of the entity's data and an abstraction of its behavior. The data (attributes) are manipulated by the methods. An object's methods are analogous to functions and procedures. Methods are used to set an attribute's value. Likewise, methods may be used to "get" an attribute's value.

One of the objectives of this project is that this model be constructed so that it is readily adaptable to future research efforts. With that objective in mind, careful consideration is given to identification and documentation of the components of the simulation. Object-oriented programming languages inherently support the specification of a body of objects that represent, as reasonably as possible, objects in the real world. By understanding and describing the objects and environment we wish to observe, we can readily translate our understanding to an object-oriented language. This object-oriented view will be maintained through each phase of the incremental development process. This view will also assist in the proper development of a graphical user interface (GUI), which will be included in later increments.

3.3 GUI Development

A recommendation of the previous research is to develop a user interface that may be used to maintain control over the simulation operation and its starting parameters. As the models become more complex with successive embellishments, we will transform the

model from a simple, Java applet-based GUI to a sophisticated, standalone application GUI. This will be accomplished only when it makes sense to do so.

3.4 Embellishments

Model embellishments will be accomplished so that any given model is clearly related to its predecessor. That is, embellishments will not radically transform any given model so that it is not recognizable from the previous model. We take this approach for two reasons: (1) If we embellish a given model too extensively and it does not perform as we expect from the analysis, it will be difficult to move backward to a previous step where the model worked properly. (2) Adding too much to the model at any given time may cause us to lose sight of our target. In other words, any drastic enhancement may inadvertently put us on a tangent that does not directly support the initial objectives.

This concludes the description of the methodology that will be used to develop the models presented in this thesis. The next chapter provides a detailed description of each model and the embellishments that differentiate them from their predecessors.

IV. Model Descriptions

4.1 Overview

This section describes in detail the design, implementation, and evolution of the various models used to simulate and ultimately visualize the growth of a breast cancer tumor in the Java environment. Not only are the structure and operation of the various models described, but model design rationale is provided as well. First, an overview of the basic 2-dimensional model and its origin are presented. More detailed design information is provided next, along with specific information on how the model is intended to function. Finally, embellishments to the 2-dimensional model are described. This process is then repeated for the 3-dimensional model.

The different model versions—both in this document and in the Java source code files—are identified using the Java package name `Tumor n`, where *n* is the specific version number of the tumor model. Each Java package contains all the Java classes necessary to implement that particular model. More information on the concept of Java classes is provided in Section 4.2.2. Both 2- and 3-dimensional model descriptions include a summary of their purpose and the expected results. Analyses of the models and tumor growth results are provided in Chapter 5.

Note that Java class and object names are shown in 10 point Courier New typeface (`JavaClass` and `javaObject`). Also, class names always start with a capital letter (`CancerCell`, `CancerTumor`, `Collection`), while an object instance of a class starts with a lowercase letter (`cancerCell`, `tumor`, `timeList`). This naming system is the standard convention used with object oriented programming and is followed throughout this document.

4.2 Two-dimensional Tumor Model

4.2.1 Overview

“CancerSim,” for *Cancer Simulation*, is the generic name used to describe any of the breast cancer simulation models developed during this project. The first model, Tumor01, is a Java applet that attempts to simulate the growth of a breast cancer tumor in unconstrained 2-D space. This model is the first attempt at a direct translation of Bassham’s previous work in MatLab™. Since Tumor01 is an applet, it can be executed from within most Web browsers. In a fashion similar to the MatLab™ model, the Java incarnation utilizes a $m \times n$ matrix to represent the tumor growth space.

The dimensions of the growth space depend on the pixel resolution chosen for the Java applet and the scale (pixel size) of the cancer cells. After simulation initialization, a numeric value representing a cancer cell is positioned approximately in the center of the `growthSpace` matrix. After placement of the first cell, the process of population doublings is initiated. A population doubling is complete when all cells under consideration are replicated. Replication is defined as follows:

1. The simulation identifies each existing cell,
2. determines its current position in the `growthSpace`,
3. randomly chooses a direction of travel, and
4. places a new cell at that position.

We use the term replication here because, with a single exception, all cells are identical. The only attribute that differentiates one cell from another in this model is the cell’s location in space.

The simulation ends when the total number of replications equals the population doublings value. The number of population doublings is a constant value “hard coded”

into the Java simulation. In subsequent versions of CancerSim, this value and other parameters can be changed at simulation runtime through a graphical user interface.

4.2.2 Class Development and Modeling Diagrams

The first effort at developing a tumor growth simulation resulted in the creation of a single class to represent the complete model. Since computer software is being used to construct the model, the term *class* is taken from the domain of object-oriented software engineering. A class is a concept that combines both descriptive data and procedural abstractions in such a way that the resulting object, an implementation of the class, reasonably describes a real world entity (Pressman, 1997: 556). Ideally, a class describes a single entity, and this concept is realized as the cancer simulation model is further developed.

Since the first model was a direct translation of Bassham's MathLab™ model, there existed the desire to quickly produce a model for proof-of-concept purposes. To reduce the overall model development time, the analysis and design phases were kept to a minimum. The result is a single class to describe Bassham's model in its entirety.

UML is fast becoming an accepted international standard for describing object-oriented concepts. UML was designed specifically to be a mechanism for standardized object modeling while being blind to any specific language implementation (Muller, 1997: 10). Figure 4-1 is a Unified Modeling Language (UML) diagram that was developed using the Rational Rose™ computer assisted software engineering (CASE) tool. In most instances, providing the particular CASE tool supports the process, a properly constructed UML diagram may be converted directly to object-oriented

programming source code such as Java, C++, or Ada. This particular feature of the CASE tool, however, was not utilized during this development effort.

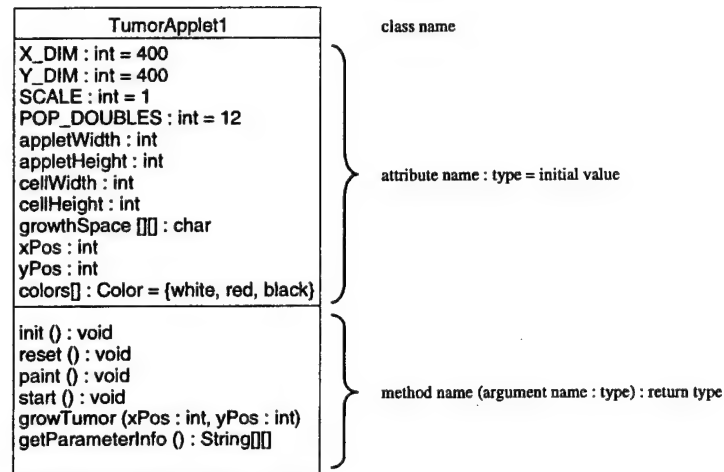


Figure 4-1: Tumor01 Class Diagram

The topmost portion of the class diagram contains the name of the class. The compartment below the class name contains each attribute name, its data type, and, if applicable, the attribute's initial value. The compartment below the list of attributes contains the programming method names and any required parameters that are passed. The data type after the colon identifies the type of object, if any that is returned after the method has completed the task for which it was designed. Note that a missing return type and the type `void` have identical meaning. Also note that a method name, its parameter list, and return types are collectively known as the method's *signature*. This term will be used in all subsequent references to a method name when it is in the context of a class diagram.

Table 4-1 lists each attribute name and a brief description. The usage contexts of the attributes and methods are included in the operational description.

Table 4-1: Attribute List (Tumor01.TumorApplet1)

Attribute Name	Description
<i>X_DIM</i>	Integer; initial width, in pixels, of the Java applet window size
<i>Y_DIM</i>	Integer; initial height, in pixels, of the Java applet window size
<i>SCALE</i>	Integer; pixel size of individual cancer cells
<i>POP_DOUBLES</i>	Integer; number of times each cell population divides
<i>appletWidth</i>	Integer; derived; actual width of the Java applet window
<i>appletHeight</i>	Integer; derived; actual height of the Java applet window
<i>cellWidth</i>	Integer; derived; represents the number of cells in the x axis of the growth space
<i>cellHeight</i>	Integer; derived; represents the number of cells in the y axis of the growth space
<i>growthSpace[][]</i>	Character matrix; derived; represents the tumor growth space
<i>xPos</i>	Integer; initial x position of the cancer cell within the growth space
<i>yPos</i>	Integer; initial y position of the cancer cell within the growth space
<i>colors[]</i>	Color array; used for painting the applet screen and cancer cells

4.2.3 Operational Description

For the early models, an applet, as opposed to an application, was chosen as the implementation vehicle because these initial versions are relatively free of embellishments. Tumor01 in particular was implemented using the Java Applet class primarily because of how quickly the model could be engineered. Before proceeding, there are a few points concerning applets that must be mentioned so that the Java source code is more comprehensible.

When a Web browser first loads an applet, there are four methods automatically called by the browser: `init()`, `start()`, `stop()`, and `destroy()`. Additionally, when a browser's refresh or reload button is pressed, the `reset()` method is called. Any of these methods may be overridden in the Java code, and Tumor01 overrides `init()`, `start()`,

and `destroy()`. In addition to these methods, `repaint()` is called automatically any time there is a need for the computer's screen to be refreshed. The method `repaint()` may also be called "manually" when necessary to update the screen at specific points in time. Again, these points are made simply because the automatic calls to some of these methods can make it difficult to trace through the Java source code.

After Tumor01 is loaded by the browser, `init()` is called and the first of two simulation initialization phases takes place. The four steps to this part of the initialization are as follows: (1) set the default applet window size to $X_DIM \times Y_DIM$ pixels, (2) readjust the `appletWidth` and `appletHeight` attributes to values that are multiples of `SCALE`, (3) set `cellWidth` and `cellHeight` to the quotients of `appletWidth/SCALE` and `appletHeight/SCALE` respectively, and (4) create the `growthSpace` matrix to the size `cellWidth` \times `cellHeight`. After these four steps, `reset()` is called manually from `init()` and all elements of the `growthSpace` are set to zero.

At this point there exists an initialized tumor growth space that is ready to receive placement of the first cancer cell. Next, `start()` is called automatically by the browser. This method determines the position of the first cancer cell within the `growthSpace` matrix, places the cancer cell, refreshes the display, and then passes the coordinates of the first cell to the `growTumor()` method.

A quick review of the Java source code reveals that a "2" is used as the value of the first cell in the matrix. When the screen is refreshed, a value of 2 in the `growthSpace` matrix will be displayed as a black cancer cell on the screen. This was done to allow visual tracking of the tumor's origin. Note that all other cells values are "1," which translate to red cancer cells when displayed on the screen.

After `start()` completes the `growthSpace` initialization, `growTumor()` is called and the `xy` positions of the first cell are passed as parameters. This method is the heart of `Tumor01`. It is here that cancer cells are replicated and tracked. Each cell's position within the growth space is tracked with a static 2-D "cell vector," an $m \cdot n \times 2$ array that is created at runtime and whose length corresponds to the number of cells available within the `growthSpace`. The number of times `cellVector` is scanned represents the number of population doublings that occur. For each cancer cell found in `cellVector`, a new cancer cell is placed in an unoccupied position in the `growthSpace`.

The direction in which a search for a free space is conducted (there are eight possible directions) is determined by a uniform random number. The random number is generated each time a new cell must be positioned. Figure 4-2 shows the possible directions of cell movement. The probability of moving in any of the indicated directions is $\frac{1}{8}$, or 0.125.

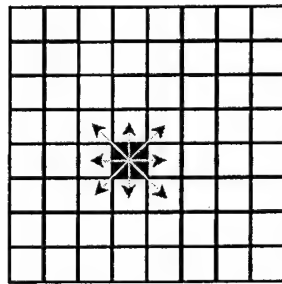


Figure 4-2: Cancer cell movement within the 2-D tumor growth space

The process of growing a tumor within the `growTumor()` method is a repetitive process that may be described as follows: After placement of the first cell as noted in the

preceding paragraph, a uniform random number is drawn on the interval [0.000, 1.000). Starting with "North" and moving clockwise through the seven remaining potential directions, the random number is compared with the values possible for each direction. If the random number is on the interval [0.000, 0.125), the cell will move North; if on the interval [0.125, 0.250), the cell will move Northeast. These checks are made until the random number fits within one the specified intervals. The Java code supporting this algorithm is shown in Listing 4-1.

Listing 4-1: Determining cell direction in 2-D space

```
// Get uniform random number on the interval [0.000, 1.000).
rNum = java.lang.Math.random();
// Use random number to determine direction of new cell's
position.
if ((0.000 <= rNum) && (rNum < 0.125)) {      //      y+ ("North")
    moveX = 0;
    moveY = 1;
}
else if ((0.125 <= rNum) && (rNum < 0.250)) { // x+, y+
    ("Northeast")
    moveX = 1;
    moveY = 1;
}
else if ((0.250 <= rNum) && (rNum < 0.375)) { // x+      ("East")
    moveX = 1;
    moveY = 0;
}
else if ((0.375 <= rNum) && (rNum < 0.500)) { // x+, y-
    ("Southeast")
    moveX = 1;
    moveY = -1;
}
else if ((0.500 <= rNum) && (rNum < 0.625)) { //      y- ("South")
    moveX = 0;
    moveY = -1;
}
else if ((0.625 <= rNum) && (rNum < 0.750)) { // x-, y-
    ("Southwest")
    moveX = -1;
    moveY = -1;
}
else if ((0.750 <= rNum) && (rNum < 0.875)) { // x-      ("West")
    moveX = -1;
    moveY = 0;
}
else if ((0.875 <= rNum) && (rNum < 1.000)) { // x-, y+
    ("Northwest")
    moveX = -1;
    moveY = 1;
}
}
```

Note that at this point the simulation has only determined a *direction* of travel for a new cell. The current cell's coordinates `xPos` and `yPos` are summed with the values representing the direction of travel: `moveX` and `moveY`. The result is a candidate position within the tumor growth space. The candidate position is then checked to see if it is currently occupied by another cancer cell. If the candidate position contains a value other than 0, the position is occupied. If a position is occupied, `moveX` and `moveY` are incremented according to a rule set supporting the current direction of travel and again summed with the original coordinates `xPos` and `yPos`. The net effect of this process is that a candidate position is searched in the same direction determined by the code in Listing 4-1.

Once a free position is found, `xPos` and `yPos` are updated to reflect the new coordinates. At this point, `growthSpace[xPos][yPos]` is set to 1. This indicates that a cancer cell now exists at those coordinates within the cancer growth space. Next, the cancer cell is effectively "added" to the `cellVector` which, again, is meant to be a mechanism for managing the collection of cells that comprise the tumor. Figure 4-3 shows the contents of `cellVector` and the positions of the indices after placement of the first cell.

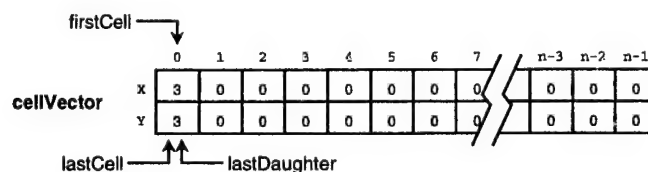


Figure 4-3: Cell vector before first population doubling

It should be noted that during each population doubling, the new (daughter) cells are logically separated within the `cellVector` by way of the `lastCell` and `lastDaughter` indices. During each scan of `cellVector`, all cells found between `firstCell` (always index 0) and `lastCell` indices will be divided. Each new daughter cell is placed *behind* the current position of the `lastDaughter` index. That is, all cells from position `firstCell` through `lastCell` cell represent the parents, and cells from `lastCell+1` through `lastDaughter` represent the daughter cells. The process for the first population doubling is detailed in Figure 4-4 and explained in the next paragraph.

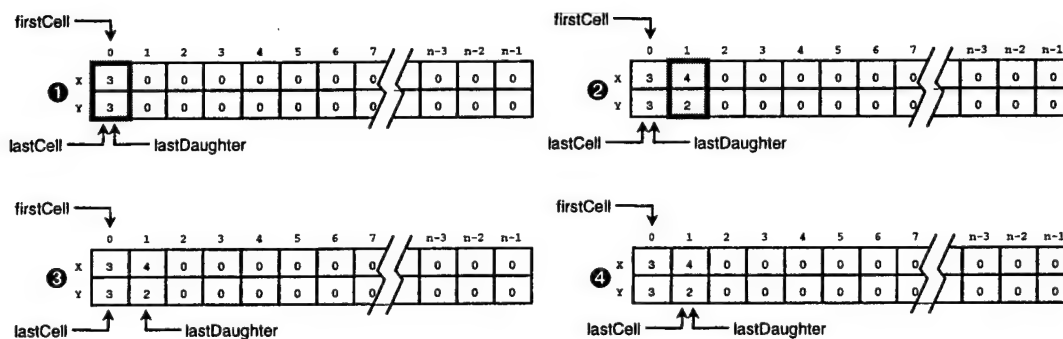


Figure 4-4: Cell vector after first population doubling (step 4)

Figure 4-4 shows the basic process behind the first population doubling, which, as described here, is independent of the details required to update the tumor growth space. (Recall that the tumor growth space is used to check for the presence of a cell at a particular location and update the tumor projection.) Every population doubling using the `cellVector` structure occurs as follows: Starting at the index identified by `firstCell`, the cell vector is scanned for parent cells and stops only after it reaches the `lastCell` index. Step 1 shows that a parent cell was found at position 0. The cell at

position 0 divides and the daughter cell is placed in the position immediately following the current position of the lastDaughter index (step 2). In step 3 the lastDaughter index is updated to point to the current “last daughter.” Since position 0 of the cell vector is also the lastCell, a complete population doubling has occurred. That is, all parent cells have divided. To finalize the process, step 4 illustrates that lastCell is updated to point to the same position as lastDaughter. This effectively promotes the daughter cells to parent cell status. Figure 4-5 demonstrates the use of the cell vector for the second population doubling, and the paragraph that follows briefly describes the process.

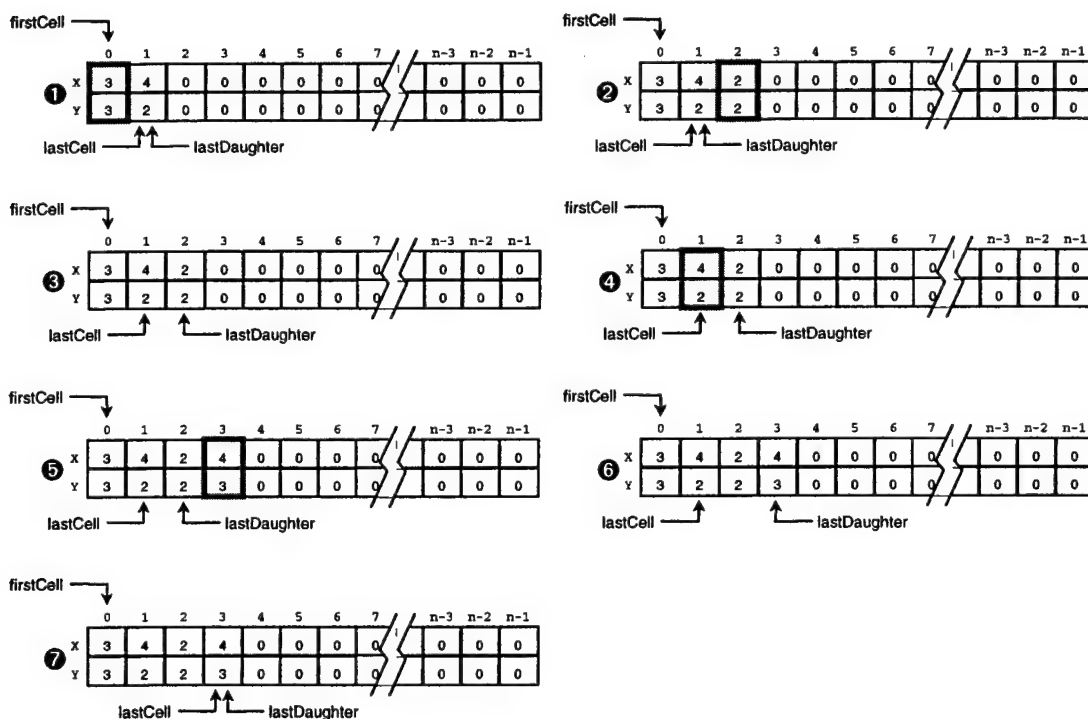


Figure 4-5: Cell vector after second population doubling (step 7)

As with the first population doubling, the second population doubling starts by scanning from firstCell through lastCell and each cell found is doubled. Step 1

shows that a parent cell was found at position 0. In step 2, the daughter of the first division is placed after lastDaughter, and in step 3 lastDaughter is updated to reflect the position of new lastDaughter. In step 4 the scanning mechanism is incremented and another parent cell is found. In step 5 the parent cell at position 1 is divided and the daughter placed at position 3. Step 6 again shows that the lastDaughter index is updated. Since the vector was completely scanned for parent cells (firstCell to lastCell), the second population doubling is complete and lastCell is updated in step 7. The process described for the first and second population doublings is repeated the number of times specified by the POP_DOUBLES attribute found in Figure 4-1.

Figure 4-6 summarizes the relationship between the major components of Tumor01. The tumor growth space is used to check for free space for a new cell. The cell vector is used to manage and track the cancer cells, and the Java applet window provides a 2-D display of the data generated in the tumor growth space.

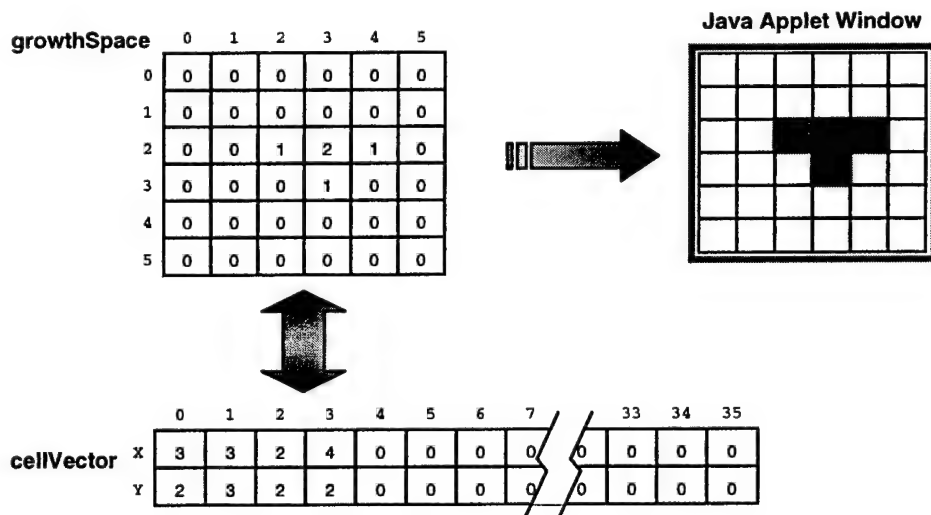


Figure 4-6: Tumor01 Object relationship diagram

4.2.4 Embellishments

4.2.4.1 Overview

The purpose of the embellishments of the 2-D model is to improve on the scalability of the model and incorporate a means to verify tumor growth *during* the process as opposed to after the tumor growth is complete. By scalability, we mean that the model can be readily adapted to more complex problems. “Readily” is certainly relative, but the intent is that this model be continuously refined so that it can be applied to more complex situations with a minimum amount of model retooling. The verification part of the embellishment is simply a nicety providing easy confirmation that the model is behaving as expected.

It was mentioned earlier in this chapter that, under ideal circumstances, an object would depict as closely as possible the real-world entity it is intended to model. Additionally, one of the objectives of this project is to attempt to model the tumor growth in such a way as to exhibit complex adaptive behavior. In simple terms, the cancer cells should be autonomous entities that are not only aware of each other, but also know how to interact with each other and their environment. If their environment changes, they have the ability to make decisions regarding their future behavior.

4.2.4.2 Model Modifications

The most obvious modification to the model—both logically and physically—is the creation of a cancer cell entity. The class `CancerCell` is the first attempt at modeling an individual breast cancer cell. At this early stage, the `CancerCell` class has only a single attribute, which is its location in 2-D space. Figure 4-7 shows the UML diagram for the model `Tumor02`.

Note that the name of the main class is changed from TumorApplet1 to CancerSimApplet. As development on the models progresses, the reader will note an improvement in the naming conventions used to describe the model, and the name change mentioned here is simply part of that improvement process.

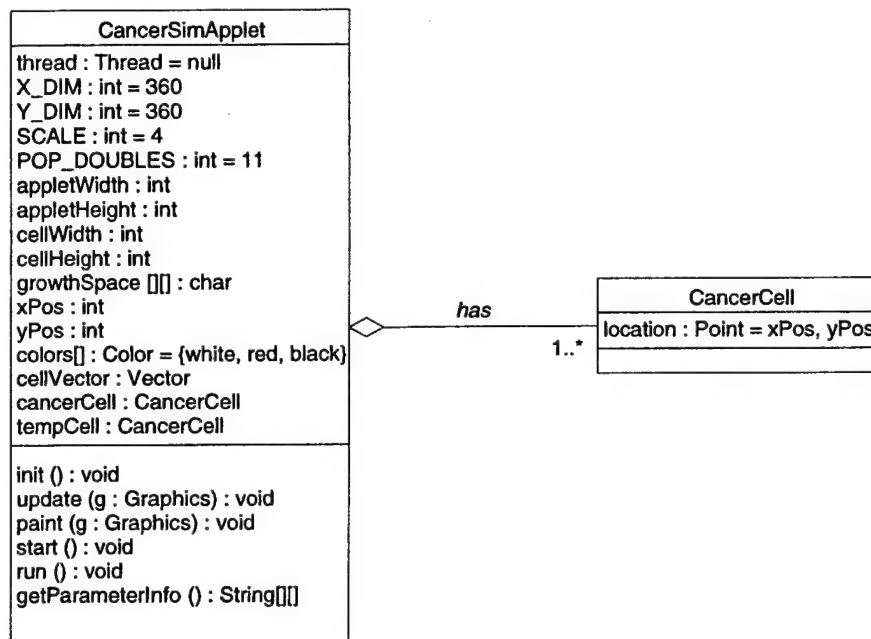


Figure 4-7: Tumor02 Class Diagram

At this point the relationship between the CancerSimApplet and CancerCell classes should be explained. The UML diagrams used throughout this document are read left to right and top to bottom. The line between the two classes in Figure 4-7 establishes the *type* of relationship and the cardinality of that relationship. The diamond attached to the left side of the connecting line indicates that the CancerSimApplet is an aggregation of the CancerCell class. Used in conjunction with the cardinality “1..*” shown on right side of the relationship, this diagram illustrates that the CancerSimApplet *has* one or

more cancer cells. Hopefully at this juncture it is apparent to the reader where further embellishments of this class relationship will lead.

Table 4-2 lists the attributes for each class in Tumor02 and provides a brief description of their purpose.

Table 4-2: Attribute List (Tumor02.CancerSimApplet)

Class Name	Attribute Name	Description
<i>CancerSimApplet</i>	<i>thread</i>	Thread; allows the applet to run within its own thread of execution
	<i>X_DIM</i>	Integer; initial width, in pixels, of the Java applet window size
	<i>Y_DIM</i>	Integer; initial height, in pixels, of the Java applet window size
	<i>SCALE</i>	Integer; pixel size of individual cancer cells
	<i>POP_DOUBLES</i>	Integer; number of times each cell population divides
	<i>appletWidth</i>	Integer; derived, actual width of the Java applet window
	<i>appletHeight</i>	Integer; derived, actual height of the Java applet window
	<i>cellWidth</i>	Integer; represents the number of cells in the x axis of the growth space; derived from applet width and scale of cancer cell
	<i>cellHeight</i>	Integer; represents the number of cells in the y axis of the growth space; derived from applet height and scale of cancer cell
	<i>growthSpace[][]</i>	Character matrix; derived; represents the tumor growth space
	<i>xPos</i>	Integer; initial x position of the cancer cell within the growth space
	<i>yPos</i>	Integer; initial y position of the cancer cell within the growth space
	<i>colors[]</i>	Color array; used for painting the applet screen and cancer cells
	<i>cellVector</i>	Vector; container to hold and manage the cancer cells
	<i>cancerCell</i>	CancerCell; object containing a point in the xy plane; represents an individual cancer cell
<i>CancerCell</i>	<i>location</i>	Point; object with attributes allowing manipulation in the xy plane

4.2.4.3 *Process Modifications*

There were several process, or model behavior, modifications made during this embellishment to improve the overall simulation performance. The first modification involved allowing the Java applet to operate in its own thread of execution. Without going into unnecessary detail, threads generally provide improved execution times and give us greater control over the timing and duration of the execution. In a nutshell, placing components of the model in separate threads allow us to start, pause, or stop the simulation as necessary. Without threads, the simulation cannot be easily interrupted.

In addition to allowing the model to run in its own thread of execution, the simple `cellVector` *array* was exchanged for a Java `Vector` class. A `Vector` can be thought of as a “container” in which we place our cancer cells. The `Vector` class includes powerful methods to manipulate virtually any object placed inside of it. Additionally, since the `Vector` only views its contents as “objects,” the `CancerCell` class can be changed extensively with absolutely no modification to the rest of the code base. This embellishment directly supports the scalability concept mentioned earlier. Despite this particular change in container objects (`Array` to `Vector`), the basic process for managing the collection of cancer cells remains the same. The real change is buried in the details of how we access the different implementations of the `cellVector` object.

Another embellishment affecting the tumor growth process is that the code in Listing 4-1 was optimized to reduce the number of checks required to obtain a cell direction. Whereas the Java code in `Tumor01` implemented a sequential check to match the random number to its interval, `Tumor02` implements a pseudo binary search that effectively reduces the expected number of checks from four to three.

Finally, minor modifications were made to “see” the tumor grow. After each new cell is placed in the `growthSpace`, a call is made to `repaint()`. Since the simulation is running in its own thread, the calls to `repaint()` are queued and then processed when it is convenient to the thread.

Even with these embellishments, the basic behavior of `Tumor02` is virtually identical to that of `Tumor01`. The most significant changes in `Tumor02` are the addition of the `CancerCell` class and the use of the Java `Vector` to store the cancer cells. These changes primarily affect how we *view* the model and how easily the model can be scaled in the future. Figure 4-8 shows the object relationship diagram with these embellishments in effect.

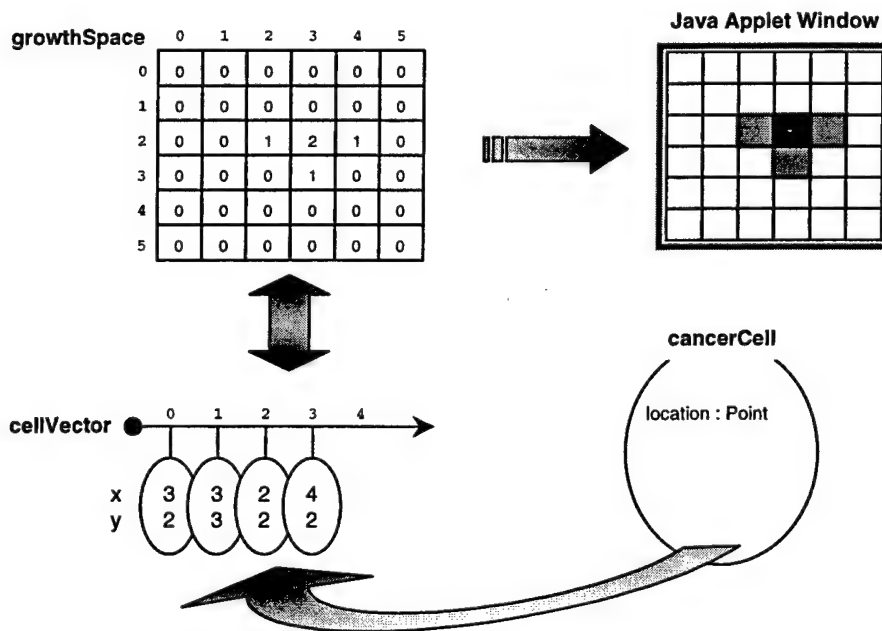


Figure 4-8: Tumor02 Object Relationship Diagram

4.2.5 Summary

The development of these initial models serves several purposes. First and foremost, it provides a simple mechanism for proof-of-concept demonstrations for the translation of Bassham's MatLab™ model to Java. In addition to proving an understanding of the basic model itself, it is expected to show that the model can be effectively implemented in a computer environment where the programming code is not native to that particular platform. In other words, it is important to show that a cross-platform, general purpose programming language such as Java can be as effective as the original model's MatLab™ code, which is optimized specifically for mathematical operations.

The second purpose of the initial models is that they provide a means to explore the possibilities of constructing a breast cancer tumor model in such a way as to emulate a real-world entity in both behavior and structure. In theory, an object-oriented programming language such as Java makes it easier to model complex real world entities. It is expected that following the object-oriented paradigm will actual make it easier to construct such a model.

Finally, the initial models are instruments with which to experiment using various techniques for visually depicting the results of the simulation. Again, it is expected that these initial Java models provide at least the same visual functionality as the MatLab™ counterpart.

4.3 Three-dimensional Tumor Model

4.3.1 Overview

The models described in this section are similar in behavior to the 2-D models presented in the previous section, with the obvious exception that a third dimension was added. Rather than constraining tumor growth to 2-D space, essentially width and height, a third dimension, depth, was added to allow virtually unencumbered tumor growth. Where the 2- and 3-dimensional models diverge, however, is in the way they are structured. The initial 3-D model attempts to capitalize on the object-oriented foundation laid during development of the 2-D models. Paragraph 4.3.2 elaborates on how these ideas are implemented.

In Section 4.3, all references to CancerSim are to models that simulate the growth of a 3-dimensional breast cancer tumor. The first 3-D model utilizes a static $l \times m \times n$ cube for the tumor growth space. The dimensions of the growth space depend on the pixel dimensions chosen for the applet and the scale (size) of the cancer cell in pixels. After initialization, a single cancer cell is positioned approximately in the center of the cube. The cell's position is tracked with a dynamic, object-oriented container (`Vector`) that is created at runtime. The number of times the container is examined represents the number of population doublings that occur. For each cancer cell found in the vector `cellVector`, a new cancer cell is placed in an unoccupied position in the cube `growthSpace`. The direction in which a search for a free space is conducted (there are 26 possible directions) is determined by a uniform random number that is generated each time a cell must be positioned. The simulation ends when the number of population doublings is complete or

the tumor growth exceeds the bounds of the `growthSpace` cube. If this condition occurs, a Java exception will be thrown (`ArrayIndexOutOfBoundsException`).

Implementation of the 3-dimensional models was accomplished using both applets and applications. Despite the variations between the methods of implementation, the 3-D models described here are expected to present virtually identical results.

Variations between these models lie mostly in efficiency of some of the procedures used to grow tumors and detect the presence of a cancer cell in a particular location. This section describes Tumor03, as well as embellishments Tumor03_1, Tumor04, Tumor04_1, and Tumor04_2.

CancerSim models Tumor03 and Tumor03_1 are the final versions using the Java applet interface. It was mentioned earlier that there are several objectives of this project. One of those objectives, which was spurred by a recommendation in Bassham's thesis, is that a graphical user interface (GUI) be built to allow user interaction with the simulation parameters. Additionally, it is a desire to allow simulation parameters as well as simulation-generated data to be saved to and retrieved from external storage devices. While a GUI is easily built using Java components supported by the `Applet` class, data transfer between the applet and secondary storage devices is strictly prohibited by Java security policies built into the language. Additionally, there are some advanced GUI components not readily supported by the current generation of Web browsers. Thus, Tumor03 and Tumor03_1 introduce a 3-D environment while retaining the basic GUI functionality found in previous versions. Tumor04 and subsequent versions carry forward the 3-D tumor growth environment while introducing more advanced user

interface components, user adjustable simulation parameters, as well as the ability to transfer data between the application and auxiliary storage.

4.3.2 Class Development and Modeling Diagrams

This version of CancerSim, Tumor03, resulted in relatively minor modifications to the two existing classes, and the creation of two additional classes. All modifications are intended specifically to support the more complex 3-D tumor growth environment.

Figure 4-9 shows the UML class diagram for Tumor03.

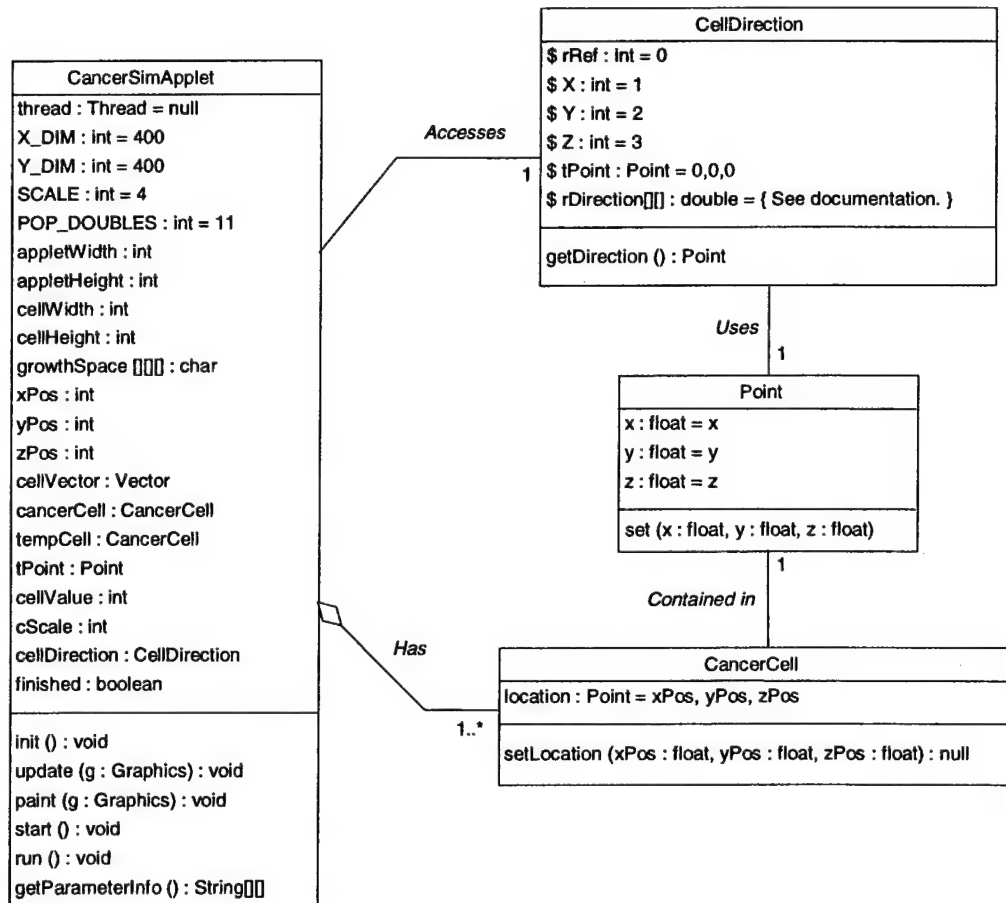


Figure 4-9: Tumor03 Class Diagram

Note the relationship between the classes depicted in the diagram. The main Java class `CancerSimApplet` *accesses* an instance of the `CellDirection` class and *has* multiple instances of the `CancerCell` class. Additionally, the `CellDirection` class *uses* a single instance of the `Point` class, and there is one instance of a `Point` class *in* each instance of a `CancerCell`. More details on modifications and construction of these classes follow. First the modifications to `CancerSimApplet` and `CancerCell` classes are described, and then the new classes and the rationale for their construction are presented.

Changes made to the `CancerSimApplet` class are primarily those required to support tumor growth in 3-D space. Where `Tumor02` contained a 2-D `growthSpace`, `Tumor03` contains a 3-D `growthSpace`. An index (`zPos`) was added to support the additional dimension, some temporary attributes were added in support of cancer cell manipulation, as well as a declaration for the new class `CellDirection`. A subtle modification with significant performance implications is the addition of a Java `Thread` attribute. This attribute is explained further in Section 4.3.3.

Changes made to the `CancerCell` class primarily involve the addition of a method to allow setting of the values for a cancer cell's location in 3-D space. As with the previous implementation of `CancerCell`, this class uses a `Point` to describe the cell's location; however, the default Java `Point` is only 2-dimensional. To support 3-D space a new `Point` was constructed. To support the object-oriented concept of encapsulation and data hiding, we do not have direct access to the attributes contained in a class. Thus, "set" and "get" methods are created so an object instance of the class can essentially access itself. We simply tell a cancer cell to set its location to the coordinates we provide in the argument list. The details of how the cancer cell sets the value of its location

attribute are not only hidden from our view, but those details are irrelevant as well. From a model development perspective, we are not concerned with how the tool *works*; rather, we are concerned with how to *use* the tool.

The `Point` class shown in Figure 4-9 provides the basic functionality required for this version of CancerSim, and it is constructed with scalability in mind. It was mentioned earlier that the initial 2-D models provided a means to explore various methods of visualizing growth of the breast cancer tumor. The Java language supports an optional application programming interface (API) called Java3D. This API includes variations on `Point` that support complex vector calculations in 3-D space. While we are not ready to implement something that complex, the path to the Java3D API must be cleared. This is done by making `Point` as compatible as possible with the Java3D counterpart. The final addition to this model is the class `CellDirection`.

The basic algorithm required for determining a direction for a new cancer cell in the 2- and 3-dimensional models is similar. As a consequence of the added dimension in the 3-D model, however, the amount of programming code required to support the algorithm is approximately three times greater. Because of the increase in code required to support a 3-D random direction, a separate class was engineered to act as a simple look-up table for determining a new cell direction. Not only does this logically isolate the operation of finding a random direction of travel, but it also simplifies and improves the readability of the Java source code. More details on the operation of the additional classes are provided in the next section. Table 4-3 lists the Tumor03 class names, attribute names and descriptions only where they changed from or added to the Tumor02 attributes identified in Table 4-2.

Table 4-3: Attribute List (Tumor03.CancerSimApplet)

Class Name	Attribute Name	Description
CancerSimApplet	GrowthSpace[][][]	Character matrix; derived; represents the tumor growth space
	Zpos	Integer; initial z position of the cancer cell within the growth space
	TempCell	CancerCell; temporary cell
	tPoint	Point; temporary point
	cellValue	Integer; value representing the depth or density of the tumor along the z-axis
	cScale	Integer; used to proportionally scale the tumor colors to provide maximum contrast for viewing
	cellDirection	CellDirection; lookup table to determine the direction of travel for a new cancer cell
CellDirectionI	finished	Boolean; used as a flag to indicate the simulation is complete
	rRef	Integer; index into rDirection
	X	Integer; index into rDirection
	Y	Integer; index into rDirection
	Z	Integer; index into rDirection
	tPoint	Point; temporary point
Point	rDirection[][]	Double; lists the random number intervals and corresponding cell direction tuple
	x	Float; represents the x coordinate in 3-D space
	y	Float; represents the y coordinate in 3-D space
	z	Float; represents the z coordinate in 3-D space

4.3.3 Operational Description

The initialization process for Tumor03 is identical to that of Tumor02: the method `init()`, which is automatically called after the browser loads the applet, proceeds to set the default applet window size to `X_DIM × Y_DIM` pixels, readjusts the `appletWidth` and `appletHeight` to values that are multiples of `SCALE`, and then sets `cellWidth` and `cellHeight` to the quotients of `appletWidth/SCALE` and `appletHeight/SCALE` respectively. Changed from previous versions is the initialization of the tumor growth

space. Since depth (z-axis) was added to this model, the `growthSpace` matrix is now created with the following dimensions:

$$\text{cellWidth} \times \text{cellHeight} \times [(\text{cellWidth} + \text{cellHeight})/2]$$

Using the default, hard coded parameters, the `growthSpace` dimensions are 100×100×100.

After initialization of the applet display and tumor growth space dimensions, `start()` is the next method automatically called by the browser. In a manner effectively identical to previous versions, this method determines the position of the first cancer cell within the `growthSpace` matrix, initializes the individual growth space cells to zero, and then places the first cancer cell at the position that was just calculated (approximately center). A minor visual difference between the 2-D and 3-D projections is that the initial cell in the 3-D version is not painted black. After placement of the first cell, the process differs somewhat from previous versions.

The purpose of a Java thread was described in paragraph 4.2.4.3. This version of `CancerSim` makes use of threads through implementation of the `Java Runnable()` interface. (See the class declaration statement in the `Tumor03` source code.) The final step carried out by the `start()` method is to create a thread instance, associate the entire applet with that thread, and then “start” the thread through use of the `start()` method. Listing 4-2 contains the code snippet from the `start()` method that creates and starts the thread. Note that the `getName()` method serves no purpose relative to the creation of a thread instance. The name of the applet may be accessed, however, for particular thread management functions, such as to see if an instance of a particular thread exists.

Listing 4-2: Thread Creation in Tumor03

```
// Set flag to indicate the tumor growth is not complete, then create an
// instance of a thread, associate this applet with the thread, and finally
// start the thread. Starting a thread automatically generates a call to
// the run() method.
finished = false;
String appletName = new String(this.getName());
if (thread == null) {
    thread = new Thread(this);
    thread.start();
}
```

The `run()` method in `Tumor03` is analogous to the `growTumor()` method in previous versions; it is here that most of the simulation processing takes place and is the heart of `Tumor03`. It is here that cancer cells are replicated, tracked and painting of the tumor projection is initiated. Each cell's position within the growth space is tracked with a dynamic Java container of the class `Vector`. An instance of `Vector` (`cellVector`) is created at runtime with a length of 512 *objects*. That is, `cellVector` can initially contain 512 of “anything.” In this simulation, `cellVector` holds one or more `cancerCell` objects. The initial length of `cellVector` will accommodate 2^8 cells, or eight population doublings, before it is required to grow. When it does grow, it adds the capacity to store 512 new objects.

After the `cellVector` object is instantiated, a new cell is created and placed in the `cellVector`. The program then enters a loop and cycles through the code in the loop the number of times specified by the `POP_DOUBLES` constant found at the beginning of the source code. The basic algorithm for the remainder of the code is provided in Listing 4-3.

Listing 4-3: Tumor Growth Algorithm in Tumor03

1. Perform the following POP_DOUBLES times:
 - A. Perform the following for each parent cell in the tumor (cellVector):
 - (1) Get a random direction of travel
 - (2) Travel in the specified direction in 3-D space until a free space is found
 - (3) Create a new cancer cell
 - (4) Assign the coordinates of the free space to a new cancer cell
 - (5) Place a 1 in the growthSpace cube at the new coordinates
 - (6) Place the new cancer (daughter) cell at the end of cellVector
 - B. If all parent cells have been replicated, update pointers to show that all daughter cells are now parent cells.
 - C. Update the screen with a count of the number of cells in the tumor
2. Set the finished flag to true
3. Paint the screen with the tumor projection
4. Stop

Note that since we are now using 3-D space for the tumor growth, there are 26 possible directions of travel for the new cell. As before, a uniform random number is drawn and compared to a list of numbers in the look-up table `rDirection` (random direction) contained in the class `CellDirection`.

Table 4-4 is representative of the `rDirection` object found in the Java code.

Table 4-4: Lookup table used in class `CellDirection`

Index	Cell Value	X	Y	Z
0	0.038462	0	0	-1
1	0.076923	0	1	-1
2	0.115385	1	1	-1
3	0.153846	1	0	-1
4	0.192308	1	-1	-1
5	0.230769	0	-1	-1
6	0.269231	-1	-1	-1
7	0.307692	-1	0	-1
8	0.346154	-1	1	-1
	(disallowed)	0	0	0
9	0.384615	0	1	0
10	0.423077	1	1	0
11	0.461538	1	0	0
12	0.500000	1	-1	0
13	0.538462	0	-1	0
14	0.576923	-1	-1	0
15	0.615385	-1	0	0
16	0.653846	-1	1	0
17	0.692308	0	0	1
18	0.730769	0	1	1
19	0.769231	1	1	1
20	0.807692	1	0	1
21	0.846154	1	-1	1
22	0.884615	0	-1	1
23	0.923077	-1	-1	1
24	0.961538	-1	0	1
25	1.000000	-1	1	1

After a random number is drawn, it is compared to each Cell Value in the lookup table, starting with the lowest index. If the random number exceeds the cell value, the index is incremented and a comparison is made with the next cell in the table. The uniform distribution is on the interval $(0, 1]$, so the random number will always be less than at least one of the numbers in the lookup table. Also note that “cell” in the context of the lookup table refers to a histogram cell from a uniform distribution. Figure 4-10 summarizes the relationship between the various objects used in this version of CancerSim.

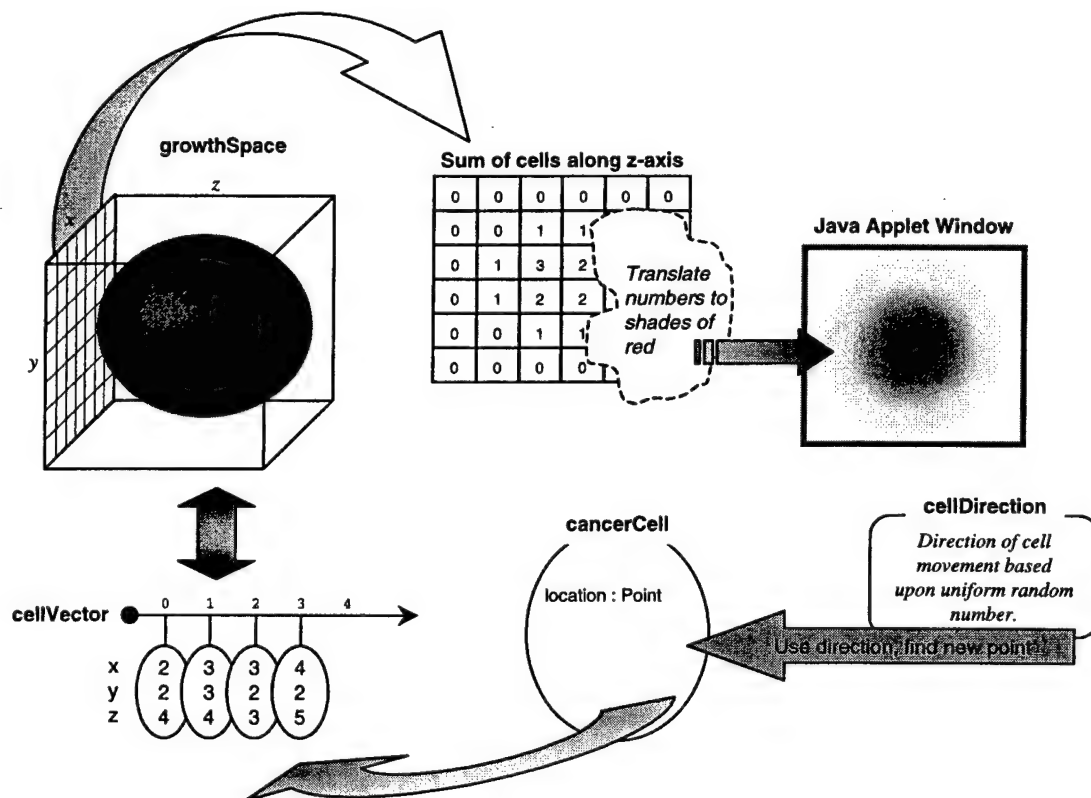


Figure 4-10: Tumor03 Object Relationship Diagram

4.3.4 Embellishments

4.3.4.1 Overview

The purpose of embellishments to Tumor03 is to continue to scale the tumor growth model. That is, provide the capability to adapt the model to more complex simulations and provide more user control over the simulation parameters. The first embellishment, Tumor03_1 is a relatively minor enhancement that allows the tumor projection to be viewed from all three axes. Tumor03_1 also represents the final implementation of the applet interface. The next embellishment, Tumor04, represents a major departure from the applet paradigm that uses Web browsers to load and start the tumor growth. Also, Tumor04 is a significant effort that takes advantages of Java Swing, an extension to the AWT that precludes the use of the applet interface. Swing components offer some advantages over their AWT counterparts and include complex interface objects that offer ease-of-use enhancements not found in the AWT. While many of the advanced components are not used here, they form a foundation for future development. Additionally, Swing offers what is referred to as a “pluggable look and feel.” In a nutshell, this feature allows identical appearance and behavior of Java applications on any computer platform.

The model Tumor04 and its variations Tumor04_1 and Tumor04_2 offer an interface that allows the user to specify certain tumor growth parameters suggested in Bassham's work. More detail on the embellishments to the 3-D model that were detailed in paragraph 4.2.3 is provided in the paragraphs that follow. First, physical modifications to the model are presented. Next, the effective changes in model behavior are described.

Finally, this section will conclude with comments on the expected behavior of these embellishments.

4.3.4.2 Model Modifications

While the capability to view the tumor projection is considered useful—or at the very least, interesting—the basic 3-dimensional model only allows a projection along the z-axis. Tumor03_1 extends the summation of the z-axis cell count to the x- and y-axes. To simplify the procedure for cycling through the projections, a system event listener was added to the model to monitor user mouse clicks.

An event listener is a Java construct that allows specific action to be taken automatically after the occurrence of a given event. In this instance, the event is the click of a mouse while the mouse pointer is positioned over the projection. This enhancement allows the user to click on the tumor window to display the next axis projection. Figure 4-11 shows the UML diagram for the `CancerSimApplet` class, which is the only class that required changes for this particular embellishment.

Note that the only change to the attributes of this class was the exchange of a `tempCell` for `axis`. The object that temporarily held a cancer cell was no longer required in the main declaration of the class, and the `axis` attribute is added to identify which of the three axes needs to be displayed. The method compartment of the class diagram shows the addition of seven mouse event methods, only one of which is used. The reason for including the six additional method signatures is explained in the Java source code for Tumor03_1.

CancerSimApplet
thread : Thread = null X_DIM : int = 400 Y_DIM : int = 400 SCALE : int = 4 POP_DOUBLES : int = 18 appletWidth : int appletHeight : int cellWidth : int cellHeight : int cellDepth : int growthSpace [] : char xPos : int yPos : int zPos : int cellVector : Vector cancerCell : CancerCell tPoint : Point cellValue : int cScale : int cellDirection : CellDirection finished : boolean axis : int = 1
init () : void update (g : Graphics) : void paint (g : Graphics) : void start () : void run () : void mouseEntered (e : MouseEvent) : void mouseExited (e : MouseEvent) : void mouseReleased (e : MouseEvent) : void mouseMoved (e : MouseEvent) : void mousePressed (e : MouseEvent) : void mouseDragged (e : MouseEvent) : void mouseClicked (e : MouseEvent) : void getParameterInfo () : String[]

Figure 4-11: Tumor03_1 modifications to CancerSimApplet class

The next embellishment to the 3-D model is realized in Tumor04. This model represents a significant departure from the previous model, not only in the “look and feel” of the user interface, but in the basic architecture as well. While we were able to continue using virtually every aspect of the previous version as the basis for this model, there were major enhancements to the existing classes and new classes were introduced. The main class, CancerSimApp, for example, no longer has many of the responsibilities it did in the previous model. In Tumor04, this class is relegated to simulation setup, such as creating an instance of the control panel and passing the default simulation parameters.

Figure 4-12 shows the UML class diagram for Tumor04. Note that the main class CancerSimApplet was renamed to CancerSimApp and the tumor growth functionality held by the CancerSimApplet class was placed in the CancerTumor class, which is new with this embellishment.

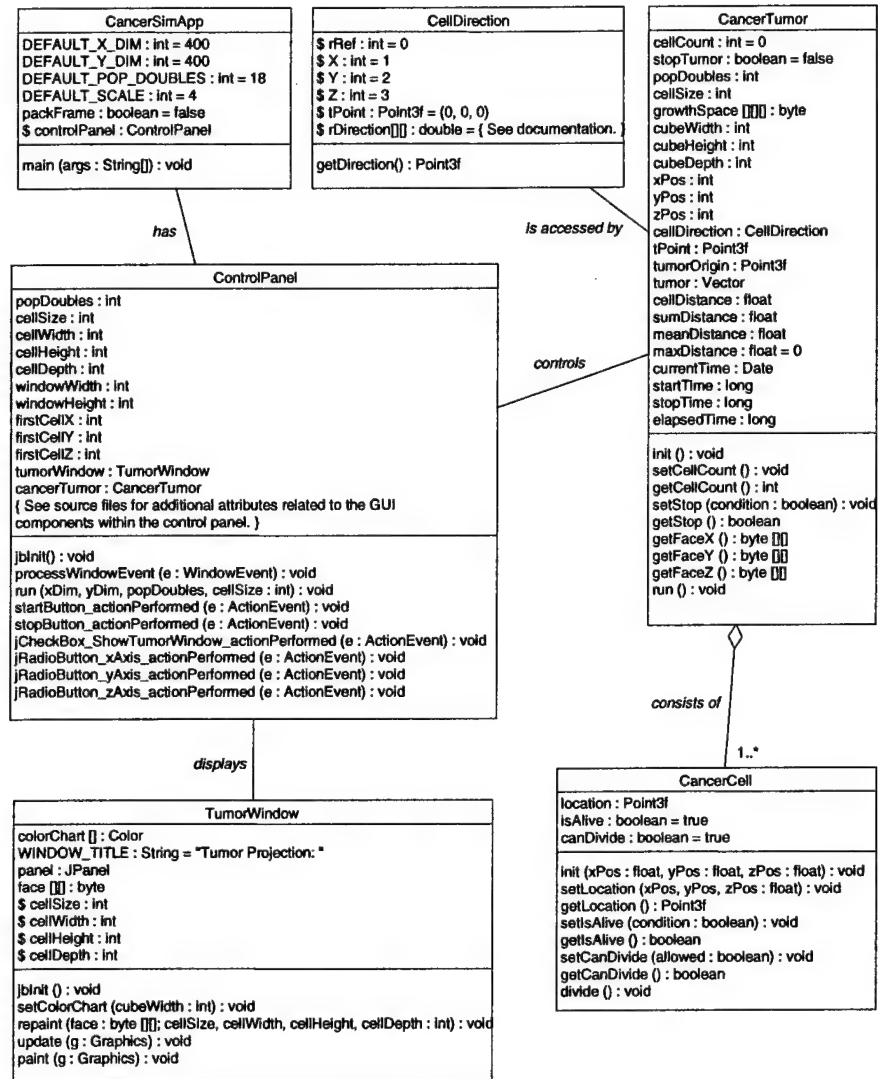


Figure 4-12: Tumor04 Class Diagram

The other obvious change to the model is the elimination of the `Point` class and the introduction of `TumorWindow`. This version of the model also completes the transition to `Point3f`, a class alluded to at the first mention of the 3-D model, and which is required for performing calculations in 3-D space.

The behavior of `Tumor04` and the data flow within the model (between objects) are explained in detail in section 4.3.4.3. It should be apparent that the introduction of the `CancerTumor` class shows the desire to move toward a model whose architecture and behavior appear to emulate that of a real world entity.

The next enhancement is an attempt to remove the overhead associated with a static 3-D growth space. Specifically, `Tumor04_1` removes the `growthSpace` cube from the `CancerTumor` class to reduce the computer memory overhead associated with simulating the growth of large tumors. In this context, “large tumor” is a tumor containing greater than 2^{20} cells. Using the current 3-D model, 2^{20} cells requires 100^3 `growthSpace`. This modification removing the growth space cube was made with very little change to the `CancerTumor` class and a diagram will not be shown here.

The final model presented in this thesis reintroduces the `growthSpace` object but alters the context of the tumor growth. `Tumor04_2` implements a subset of the work presented by Palmari, *et al* (1997). The tumor growth in `Tumor04_2` is based on time and each cell goes through three distinct phases before entering mitosis. This model again is relatively more complex than previous models; thus, a diagram is provided for clarification. The UML class diagram for `Tumor04_2` is provided in Figure 4-13.

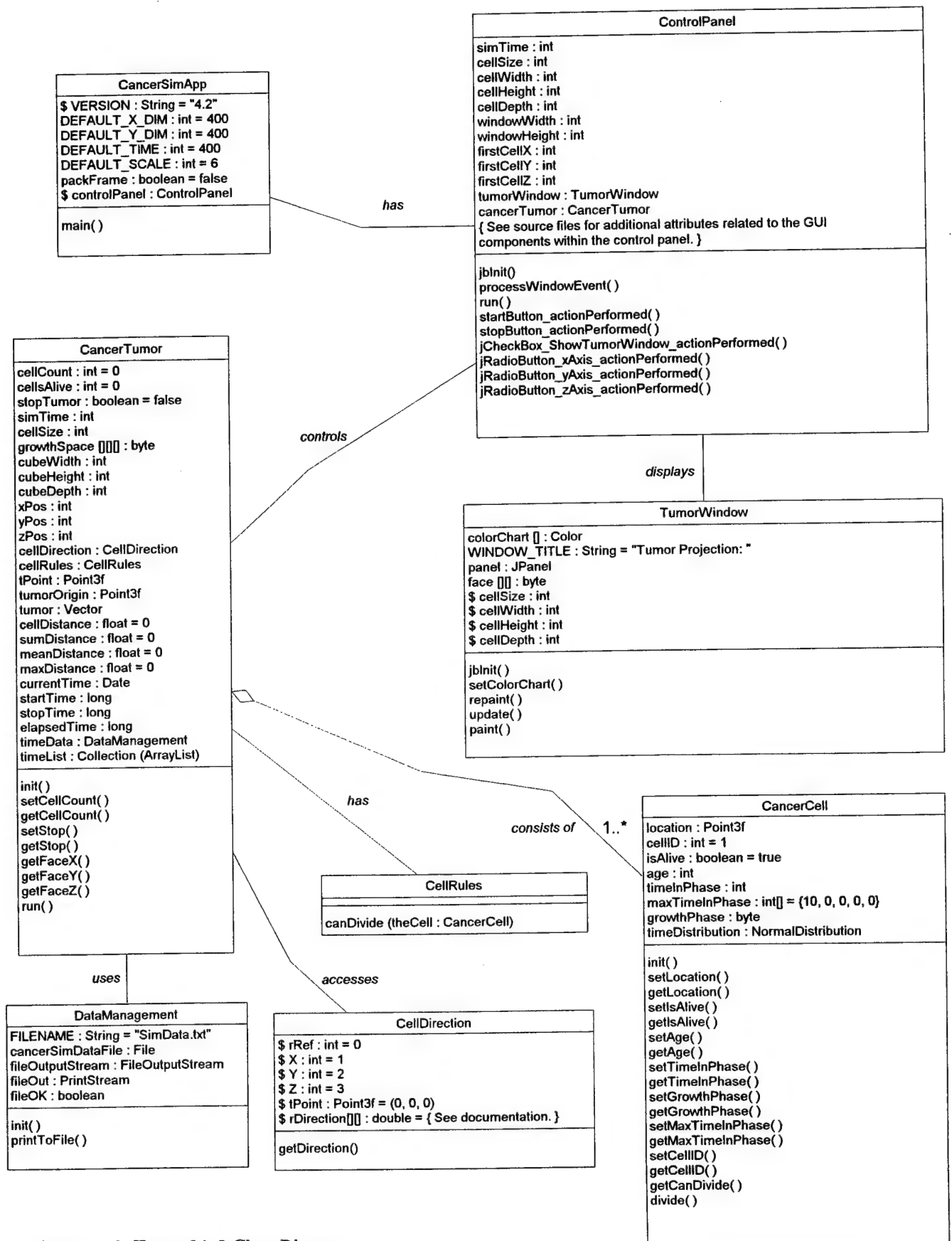


Figure 4-13: Tumor04_2 Class Diagram

For the final embellishment, one class remains unaltered: `CellDirection`. From the perspective of modification required to implement the essence of the final embellishment, `TumorWindow` is unaltered as well. There was a programming error corrected in `TumorWindow`, however, and this will be addressed in the discussion that follows. Some classes required a broad range of modifications—from minor to major—in order to accommodate the tumor growth context change from population doublings to time. The classes requiring modifications include `CancerSimApp`, `ControlPanel`, `CancerTumor`, and `CancerCell`. To complete the embellishment, two classes are added: `CellRules` and `DataManagement`. The modifications made to `TumorWindow` are discussed first, then changes supporting the essence of this final embellishment will be described.

A programming logic error was discovered in the `TumorWindow` class for models `Tumor04` and `Tumor04_1`, which resulted in a minor code correction. When running `Tumor04` with a small number of population doublings, below 10 for example, the contrast on the resulting tumor projection was so shallow that the tumor was difficult to view. This same logic error caused a more severe error in `Tumor04_1`, which completely prevented the projection from being displayed. The reader will note a subtle change in the `face[][]` attribute the `TumorWindow` class between Figure 4-12 and Figure 4-13. The attribute type is changed from `byte` to `integer`. When the cell density along the axis of projection exceeded 128, the next number entered into the projection was -127. This resulted in incorrect contrasts. The fact that the Java primitive type `byte` is signed, with valid values on the interval [-127, 128] was overlooked. Changing the type from `byte` to `integer` corrected this. The second correction improved the poor contrast of small

tumors. Previously, the range of colors used to present the contrast was based on the maximum possible tumor density. For example, a growthSpace of 150^3 would provide for a maximum density of 150 cells along any axis. While this maximum was used to calculate the contrast, the tumor growth rarely reached the maximum size allowed; thus, poor contrast for small tumors. To correct this, the code in the class TumorWindow was modified to use the actual tumor density for the contrast calculation. This change results in high contrast even for single-celled tumors. The code snippet in Listing 4-4 shows how the contrast is calculated. It is presented at this time because understanding this particular component is required if changes in projection contrast or color are to be made later.

Listing 4-4: Determining the maximum contrast for a tumor projection

```
// Create a table of Java Color objects by determining the range of colors to
// use, and then assigning a color value to each possible value in the range.
// Inputs: Tumor density + 1.
// Return: n/a
public void setColorChart(int density) {
    int cScale = (256/density); // Valid color range is 0-255.
    colorChart = new Color[density]; // Create array of length = density.
    // Fill color chart with color values that are multiples of cScale. The net
    // effect is the use of the full range of the color scale for max contrast.
    // We are using shades of white to represent the tumor density, so there is
    // no need to differentiate between the red, blue, and green arguments
    // passed to the Color() constructor. Instead, we will use a single,
    // identical value for all arguments.
    int c = 0; // Initial rgb value. Color(0,0,0) ==> black.
    // Construct the color chart. No cells in the projection will show as
    // black. Max cells in the projection will show as white. The number of
    // cells between 0 and Max will be displayed as shades of gray from black
    // to white by steps of cScale.
    // For example:
    //     density = 8 (0-7 cells)
    //     cScale = 256/8 = 32
    //     colorChart = {{0,0,0},{32,32,32},{64,64,64},{96,96,96},{128,128,128},
    //                   {160,160,160},{192,192,192},{224,224,224}}
    for (int i = 0; i < density; i++) {
        colorChart[i] = new Color(c, c, c);
        c += cScale;
    }
}
```

The CancerSimApp and ControlPanel classes are the first two classes requiring changes as a result of the shift from population doublings to time as the mechanism to control tumor growth. All changes to these two classes are minor display labeling and attribute name changes. Where we had DEFAULTT_POP_DOUBLES, for example, this is changed to DEFAULT_TIME; and popDoubles is changed to simTime. These subtle changes are representative of the changes to classes CancerSimApp and ControlPanel.

More extensive changes were made to the CancerTumor and CancerCell classes.

The CancerTumor class added four and changed one attribute:

1. cellsAlive was added. This attribute tracks the number of living cells in the tumor.
2. cellRules is an instance of a new class. This object contains the behavioral rules for a cancer cells.
3. timeData also is an instance of a new class, DataManagement. This class contains the method necessary to write data to a disk file.
4. timeList was added to track time stamps during tumor growth.
5. popDoubles was renamed to simTime to account for the context change mentioned earlier.

The CancerCell class contains the most significant changes to the model. This class added six attributes and corresponding “set” and “get” methods for five of the six new attributes:

1. cellID gives each new cancer cell a unique identifier.
2. age represents the age of the cancer cell, in time units, since it was created.
3. timeInPhase is similar to age. It represents the amount of time spent in a particular phase.
4. maxTimeInPhase indicates the maximum amount of time allowed in each of four distinct cell phases.
5. growthPhase tracks the growth phase of the cancer cell.
6. timeDistribution is an instance of NormalDistribution, a drop-in object from OpsResearch.com (<http://www.opsresearch.com/OR-Objects/index.html>). This object creates a normal distribution with a mean of μ and a standard deviation of σ . It is used to determine maxTimeInPhase.

The methods required to access these attributes are not described here. Section 4.3.4.3 describes the behavioral aspects of the model changes described above.

4.3.4.3 Process Modifications

The tumor growth and projection processes for embellishments Tumor03_1, Tumor04, Tumor04_1, and Tumor04_2 are summarized in Listing 4-5.

Listing 4-5: Tumor growth process for 3-D embellishments

-
1. Initialize the simulation growth space, projection window, and cell size
 2. Place the initial cell in the growth space (if applicable)
 3. Perform the following steps the number of times specified by the population doubles or simulation time parameters:
 - A. Perform the following for each parent cell in the tumor:
 - (1) If time-based, go to (a), else go to (2)
 - (a) If in Mitosis, go to (2)
 - (b) If time in phase exceeds maximum time in phase, go to next phase
 - (c) Go to step 3.A.
 - (2) Get a random direction of travel
 - (3) Travel in the specified direction in 3-D space until a free space is found
 - (4) Create a new cancer cell; if time-based, randomly determine the amount of time to spend in growth phase G1
 - (5) Assign the coordinates of the free space to a new cancer cell
 - (6) If growthSpace cube is used, place a 1 at the new coordinates
 - (7) Place the new cancer (daughter) cell at the end of cellVector
 - B. If all parent cells have been replicated, update pointers to show that all daughter cells are now parent cells
 - C. If applet-based, update the screen with a count of the number of cells in the tumor
 4. Set the finished flag to true
 5. Paint the screen with the tumor projection. If model Tumor03_1 or later, allow projections from all three axes
 6. Stop
-

Model Tumor03_1 is identical in operation to Tumor03 except that a projection from any of the three axes may be displayed. This is accomplished by adding the Java mouse event listeners that were described briefly at the beginning of paragraph 4.3.4.2. When the user clicks the mouse button anywhere in the projection window, the projection is changed from one axis to the next. The projections may be cycled from the x-axis, to the y-axis, to the z-axis, and then back to the x-axis.

This concludes the extent of the process changes characterizing the first embellishments to the basic 3-D model. The next paragraphs describe the process

changes inherent in the transition from an applet to an application, along with the applicable behavioral embellishments.

After transitioning from the Java Applet class to a standalone application, we now have more control over the simulation operation, including the parameters used in defining the tumor growth environment. Starting with model Tumor04, the user is presented with a GUI control panel containing common user interface (UI) components: push buttons, check boxes, radio control buttons and text boxes. Tumor growth is initiated by pressing the start button, and pressing the stop button interrupts the growth. To display the tumor projection, a check box is selected; pressing one of three radio buttons changes the axis of view. The control panel is shown in Figure 4-14. The parameters section of the control panel contains the default settings for Tumor04_1.

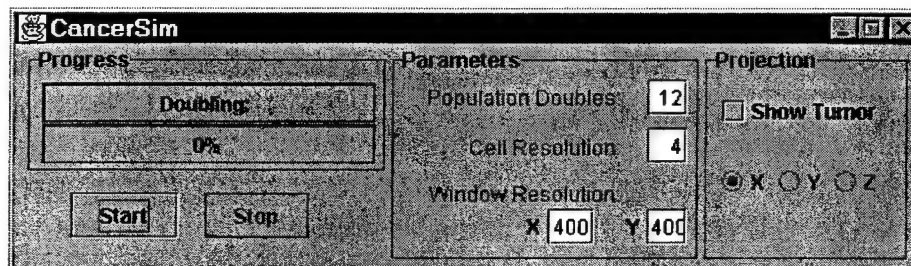


Figure 4-14: CancerSim Control Panel (Tumor04, Tumor04_1)

After tumor growth is initiated by pressing the start button, the behind-the-scenes growth process is identical to that of Tumor03 and Tumor03_1. The only variation is that timestamps are taken at the beginning and end of each population doubling, and the difference is sent to the Java session's DOS window as a number. These time stamps, which are presented in seconds, represent the elapsed time of each population doubling.

Once the simulation is complete, the start button is enabled to allow for another simulation run.

The process for Tumor04_1 produces results identical to that of Tumor04. While the starting procedures and results are identical, the underlying process is however somewhat different. Because the growth space cube (growthSpace) described in paragraph 4.3.4.2 was removed from this model, Tumor04_1 is expected to take more time to complete each population doubling; thus, the overall simulation will take longer to achieve the same results achieved in Tumor04. Without the growth space cube for cell collision detection, the cell vector is scanned from element 0 to n , where n is the number of cells in the tumor during any given cell doubling. The relationships between the objects in Tumor04_1 are represented in Figure 4-15. The components in this figure should be compared to Figure 4-10.

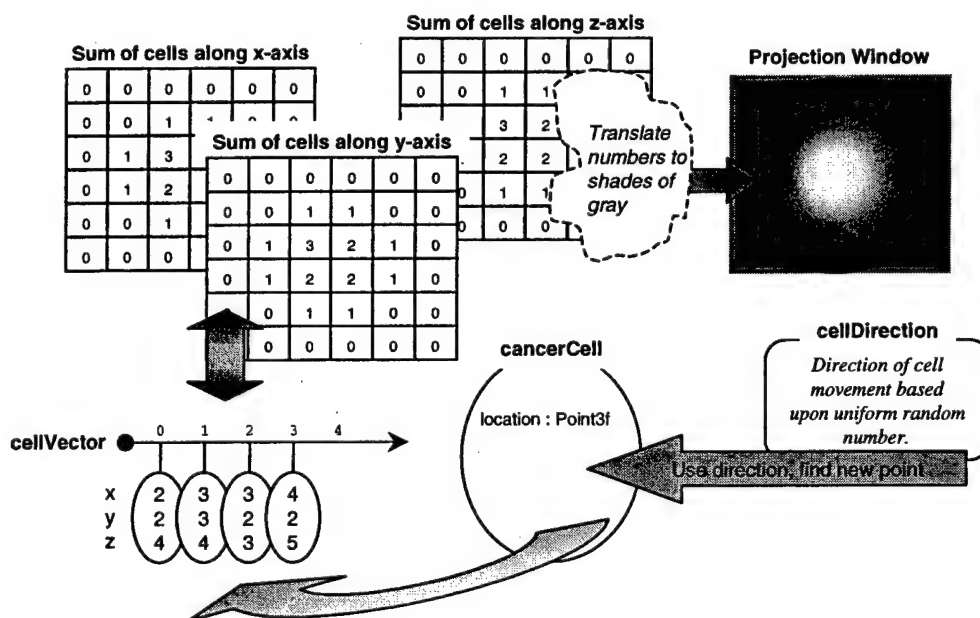


Figure 4-15: Tumor04_1 Object Relationship Diagram

One of the objectives of this thesis is to model the somewhat atomic behavior of individual cancer cells. This behavior, in general, is expected to influence the shape and rate of growth of a tumor. Tumor04_2 returns the growth space cube to the `CancerTumor` class, but the simulation context changes from population doubling-based to a time-based simulation. In models Tumor04 and Tumor04_1, the `cellVector` object is scanned from index `firstCell` to `lastCell`, and each cell “touched” subsequently divides (see Figure 4-5). In Tumor04_2, each pass through the entire `cellVector` represents the passage of one simulation time unit. The `cellVector` object is scanned in a manner identical to the previous models, but each cell is not automatically doubled; rather, the amount of time the cell has spent in its current growth phase is checked. Only after the cell has reached mitosis, phase M, is the cell doubled. The behavior for this new set of rules is outlined in Figure 4-16. Notice that the amount of time spent in each phase is determined by a random draw from a normal distribution. The entire process can be described as follows:

When the first cell is created, its age and time in phase (TIP) are set to zero. The maximum time that the cell is allowed to remain in the first phase (G1) is a random draw as shown in the diagram. Once the growth process starts, each cell in the `cellVector` is examined, its age and TIP incremented, and then the TIP is examined to see if it exceeds that maximum TIP. If so, the cell is “told” to enter the next phase. This process repeats for each cell in `cellVector` the number of times specified by the user at the start of the simulation.

Due to the fact that the parent cells are no longer divided at what was essentially the same time, it is expected that the resulting tumor’s shape will be less spherical than tumors in previous models. Additionally, the time spent in each phase leading up to

mitosis acts as a time delay, so growing larger tumors should take longer than with models prior to Tumor04_1. The paragraph that follows summarizes the models described in this chapter.

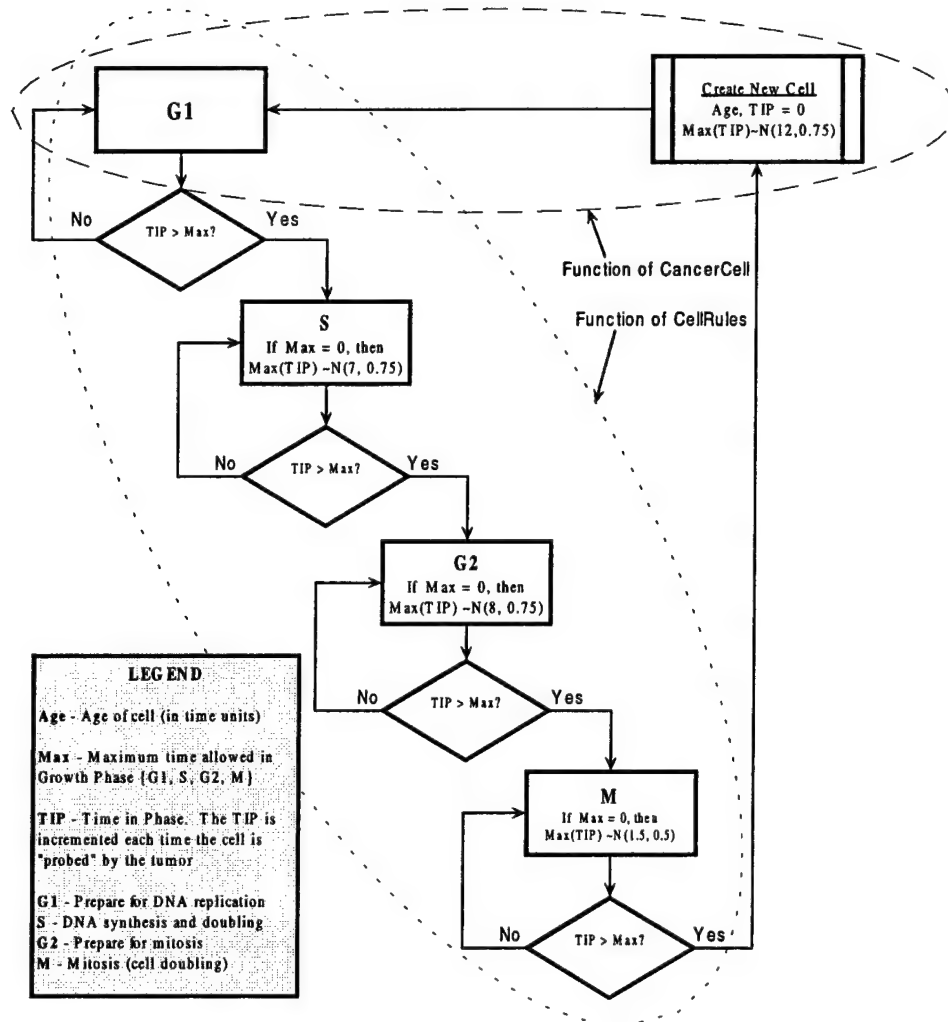


Figure 4-16: Simulation flow chart (Tumor04_2)

4.3.5 Summary

Seven models were developed in this effort. The fundamental ideas behind the design of each model were explained, the architecture of each model was presented, and the expectations described. Table 4-5 summarizes the models discussed in this chapter and provides brief comments on the characteristics that differentiate them from the other models. The chapter that follows shows the results of running each model. Ultimately, we determine which model best meets the objectives described at the beginning of this thesis.

Table 4-5: CancerSim model summary

Model	Java Classes	Remarks
Tumor01	TumorApplet1	Snake-like 2-D tumor growth; Java applet implementation
Tumor02	CancerSimApplet CancerCell	Correct implementation of 2-D model presented by Bassham; tumor growth delayed for viewing purposes
Tumor03	CancerSimApplet CancerCell CellDirection Point	Implementation of Bassham's 3-D model; density projection enhanced through use of red shading; use of color produced visual side effect causing projection to appear 3-D
Tumpr03_1	CancerSimApplet CancerCell CellDirection Point	Changed color shading from red to gray, mitigating visual side effect; added capability to view all three projections; user changes view by clicking mouse on tumor projection
Tumor04	CancerSimApp CancerTumor CancerCell CellDirection ControlPanel TumorWindow	Removed Applet class and implemented as standalone application; shifted majority of work to cancer tumor object; created GUI control panel that allows user to start and stop growth and change growth parameters; added projection window that may be turned on or off and placed anywhere on the computer desktop; timestamps are output to a DOS box to show the elapsed time for each population doubling
Tumor04_1	CancerSimApp CancerTumor CancerCell CellDirection ControlPanel TumorWindow	Removed 3-D growth space object (cube) from CancerTumor class and performed cell collision detection strictly using the cell container; growth slowed significantly over previous model
Tumor04_2	CancerSimApp CancerTumor CancerCell CellDirection ControlPanel TumorWindow CellRules DataManagement	Returned 3-D growth space object (cube) to CancerTumor class; added class to control cell behavior by defining rules for the cell growth cycle; added class to facilitate data collection by saving data to an external file

V. Analysis and Recommendations

5.1 Overview

This chapter presents the results of the models developed in the previous chapter. Execution of each model will be described and the output shown. Comments on the actual results versus the expected results are provided, and comparisons with Bassham's work are made when possible. Where appropriate, mathematical analysis of the results is accomplished. Finally, recommendations on areas of improvement to these models are made.

5.2 Two-dimensional Tumor Models

We mentioned in the previous chapter that the development of the initial models serves several purposes, one of which is to provide a proof-of-concept demonstration for the translation of Bassham's MatLab™ model to Java. That is, initially, we wanted to show that Bassham's model could be replicated in the Java environment. To make this comparison, we run a simulation using the first model, Tumor01, and provide the following initialization parameters:

- ✓ X_DIM = 400
- ✓ Y_DIM = 400
- ✓ SCALE = 1
- ✓ POP_DOUBLES = 12

These parameters provide a 400^2 growth space and 2^{12} cancer cells at a resolution of one pixel. Using these parameters, the simulation was run using the latest versions of two of the more popular Web browsers: Microsoft Internet Explorer and Netscape® Navigator. Precise time measurements were not made, but the tumor growth completed successfully in just a few seconds under both browsers.

The first model was successful in providing a limited simulation of the growth of a breast cancer tumor in 2-D space, and the results were interesting. The results were so interesting, in fact, that it took some time to determine that there was actually an error in the code used for the simulation.

After placement for the first cell, the process of a population doubling begins. The growth space is searched and the first (only) cell is found and it divides. For the second population doubling, two cells exist; therefore, two cells divide. The error in the algorithm prevents *each* cell in the population from doubling as designed; rather, the *last* cell placed is the one cell that is always divided. The intended process is shown in Figure 5-1, while the erroneous process is shown in Figure 5-2.

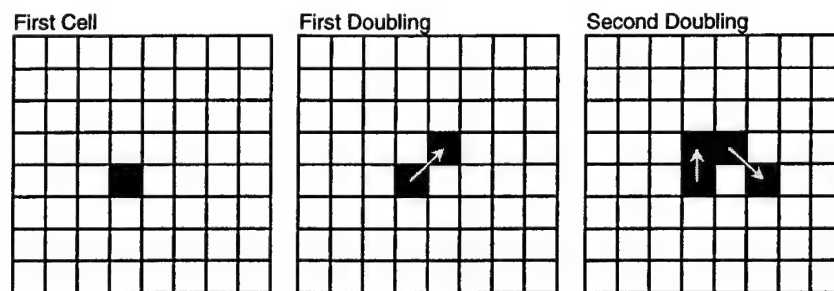


Figure 5-1: Tumor01 population doubling (expected)

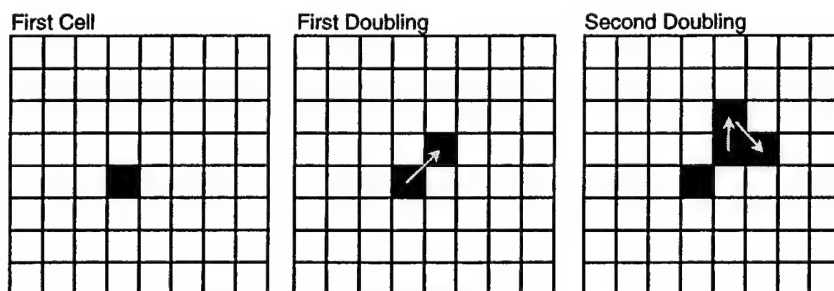


Figure 5-2: Tumor01 population doubling (actual)

The result of tumor growth using model Tumor01 is shown in Figure 5-3.

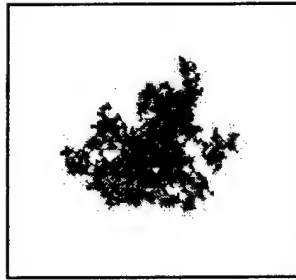


Figure 5-3: Tumor01 results

The irregular and interesting shape from this simulation is caused by the snake-like growth depicted in Figure 5-2. The logic error discovered in Tumor01 is ultimately corrected in Tumor02.

The parameters and behavior of the 2-D model were changed slightly so that growth could be observed. The specific changes made to Tumor02 are described in the previous chapter. The default parameters for this simulation are as follows:

- ✓ X_DIM = 400
- ✓ Y_DIM = 400
- ✓ SCALE = 4
- ✓ POP_DOUBLES = 11

These parameters provide a 100^2 growth space and 2^{11} cancer cells at a resolution of four pixels. Using these parameters, the simulation was again run using the Web browsers identified earlier. For demonstration purposes, a 10-millisecond time delay was introduced into the tumor growth process. This delay occurs after each cell division and prior to a screen update. Because the screen update commands are queued, the screen is not necessarily updated after each cell divides. Even with this delay in place, the entire simulation is completed in 30 seconds or less.

After confirming that the model appeared to be performing as expected, it was run again using parameters similar to those used by Bassham's Tumor5 model: 360, 360, 9, 8. These parameters provide a 40^2 growth space and 2^8 cancer cells at a resolution of nine pixels. Note in Figure 5-4 the similarities between Tumor02 and the results obtained by Bassham. The algorithm itself for Tumor02 was again compared to Bassham's 2-D algorithm and the Tumor02 implementation appears to be correct. This assertion is supported by the visual comparison.

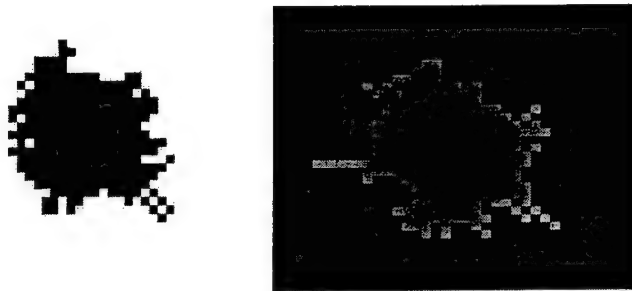


Figure 5-4: Tumor02 results [left] compared to Bassham's Tumor5

We now make the assumption that Bassham's 2-D tumor growth algorithm is correctly implemented, and we proceed with the results of the 3-D algorithm.

5.3 Three-dimensional Tumor Models

The evolution of the tumor growth model advances using the same algorithm shown in Figure 5-2, except that it is applied to 3-D space. That is, instead of limiting a new (daughter) cell to movement in the x-y plane, a daughter cell may now be placed along the z-axis as well. In addition to the 3-D growth, variations in color were added to enhance the visual representation of the tumor.

The default simulation parameters using the Tumor03 model are as follows:

- ✓ X_DIM = 400
- ✓ Y_DIM = 400
- ✓ SCALE = 4
- ✓ POP_DOUBLES = 18

These parameters provide a 100^3 growth space and 2^{18} cancer cells at a resolution of four pixels. Using these parameters, the simulation was again run using the Web browsers identified earlier. Despite eight additional population doublings, the simulation continues to run rather quickly, consistently turning in times of under 30 seconds. Time, however, is only a minor consideration. Of greater importance is that the first of the 3-D models under Java achieved 2^{18} cells where the MatLab™ counterpart achieved 2^{12} cells before running into difficulty.² The results of Tumor03 are shown in Figure 5-5.



Figure 5-5: Tumor03 results

² Using the Student Edition of MatLab™.

Not readily apparent here is an unintended side effect of the choice of color and shading. When viewed using the Java Applet Viewer or a Web browser, the tumor result has an uncanny 3-dimensional appearance, despite the fact that is a density projection and should appear *flat*, or 2-dimensional. Also note the nearly perfectly spherical shape of the tumor. Given that our algorithm chooses a direction of travel based upon a uniform distribution without consideration of body tissue or the density of co-located cells, this is expected.

Despite the interesting 3-D artifact from the choice of colors used in the projection, the visual result deviates from what was intended; therefore, the use of colors was altered for the next model, Tumor03_1.

The results of Tumor03_1 are identical to that of Tumor03. Recall from the previous chapter that Tumor03_1 merely incorporated minor changes to allow the user to view the projection from any of the three axes, and it also changed the color from shades of red to gray, as alluded to in the previous paragraph. The latter modification was intended to mitigate the 3-dimensional effect of Tumor03 and provide a more x-ray or mammogram-like appearance.

Figure 5-6 contains the results of the modifications applied to Tumor03. The default parameters for Tumor03_1 are identical to the defaults for Tumor03. Note the uniformity of the projections. Again, given that the direction of travel continues to be randomly drawn from a uniform distribution, this is expected.

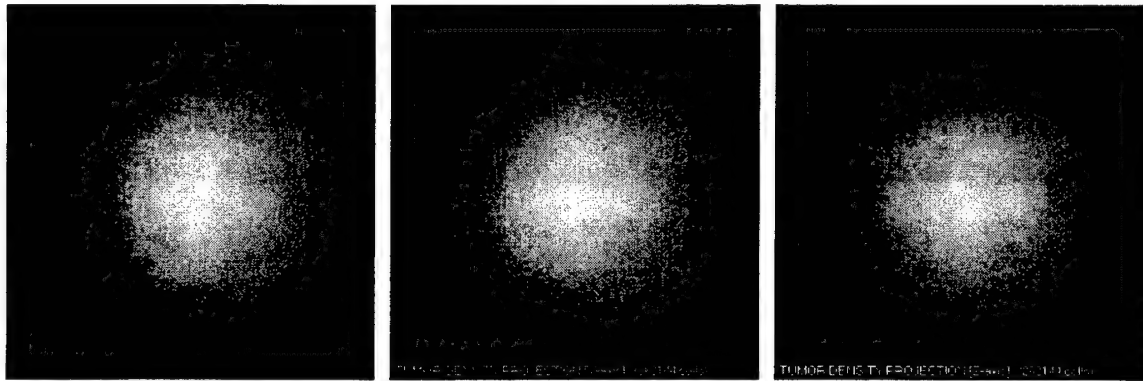


Figure 5-6: Tumor03_1 density projections along the x-, y-, and z-axes

The use of multiple projections in Tumor03_1 provided a useful transition to the next phase: implementation of a graphical user interface without reliance on the Java Applet class. The intention was to transition to a standalone Java application utilizing GUI components and the Tumor03_1 framework. As a reminder, the desire to move away from the use of Web browsers, and thus Java applets, as an implementation vehicle for the more complex models is borne of the inconsistencies in the support of Java Swing components. The development of Java and its enhancements has far outpaced browser development and support. As we mentioned in the previous chapter, Java Swing components are considered essential for developing a sophisticated interface to the simulation models.

The transition from the applet environment to a standalone Java application was successful in that the model required only minor rework and the addition of two classes for the GUI components: one class for the control panel and another class for the tumor projection window. Figure 5-7 shows the simulation environment for Tumor04. Absent is the tumor projection window.



Figure 5-7: Tumor04 GUI and DOS window

Note that this version of the model offers the capability to alter the four simulation parameters that were previously hard coded in the simulation. For the sake of a visual comparison with one of Bassham's models—specifically model Tumor3d—the simulation was run with the following parameters:

- ✓ X_DIM = 200
- ✓ Y_DIM = 200
- ✓ SCALE = 8
- ✓ POP_DOUBLES = 8

These parameters provide a 25^3 growth space and 2^8 cancer cells, which is identical to the test presented in Bassham's work. There are two points to be made concerning this model. The first is that the transition from the Java Applet class to a standalone application gave us a tremendous boost in performance. Using the above

parameters, for example, a simulated 3-dimension tumor was grown in approximately one second. This reduction in the amount of time it takes to grow a tumor proves useful for larger models. The second point is that we again achieved startlingly similar, yet expected, results when compared to Bassham's Tumor3d. Figure 5-8 is the DOS window output from a sample run of Tumor04.

```
.CancerSimApp
Press Ctrl+C to terminate the application...
=====
Tumor growth started with 8 population doublings.
Cell resolution: 8 (pixels)
Window resolution: 25 x 25 (cells)
First cell positioned at 12, 12, 12
=====
Doubling 1 elapsed time: 0.015
Doubling 2 elapsed time: 0.047
Doubling 3 elapsed time: 0.062
Doubling 4 elapsed time: 0.078
Doubling 5 elapsed time: 0.078
Doubling 6 elapsed time: 0.093
Doubling 7 elapsed time: 0.109
Doubling 8 elapsed time: 0.14
Maximum distance from tumor origin: 7.071068 units.
Mean distance from tumor origin: 3.69658 units.
```

Figure 5-8: Tumor04 sample DOS box output

The output to the DOS window provides several pieces of information that prove useful to understanding if the model is operating as expected. First, we note that the parameters used to start the simulation are echoed to the DOS window. Second, the output shows the position of the first cell. Recall that the location of the first cell in 3-D space is derived from the window and cell resolutions. Third, in this example, the elapsed time from each population doubling is calculated and displayed. Finally, the example above makes use of the Java3D library that was discussed in the previous chapter. This library includes methods for calculating distances in 3-D space. We see above that some of the library methods were used to calculate maximum and mean tumor radius.

Figure 5-9 shows a sample of the visual output from Tumor04 as compared to Bassham's Tumor3d. The titles of the projections have been clipped due to the small size of the projection window. Also note the terminology used in Bassham's models. Where Tumor04 uses projection views of X, Y, and Z, Tumor3d uses "front," "overhead," and "side." Projections from the Tumor04 and Tumor3d models are read in the same manner: lighter areas of the projection indicate greater density.

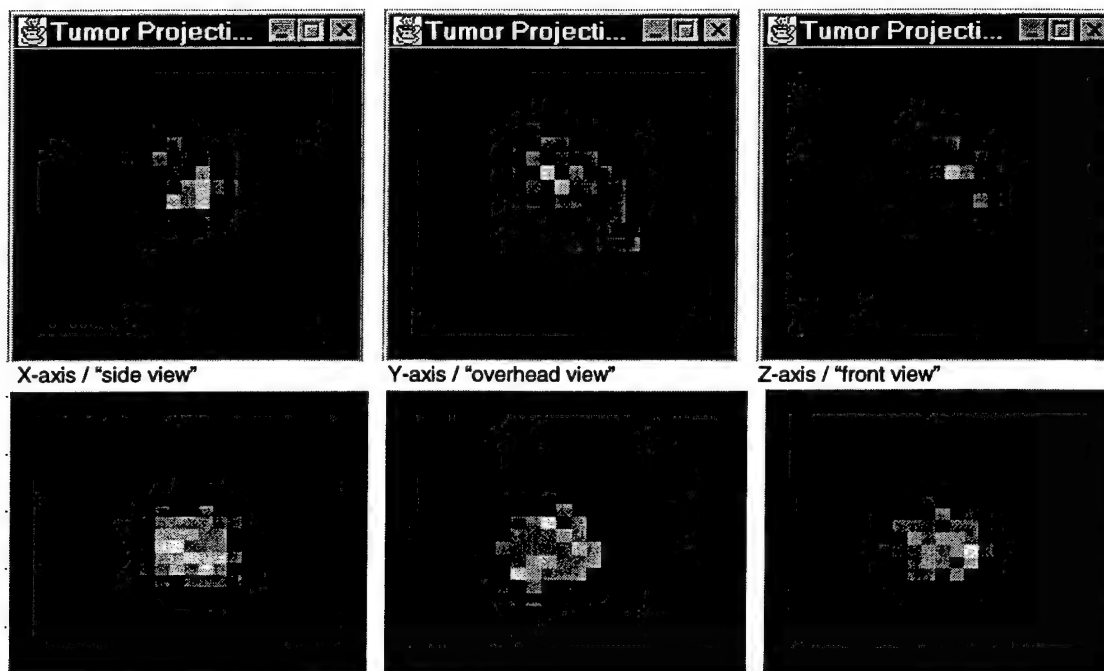


Figure 5-9: Tumor 04 [upper projections] and Bassham's Tumor3d [lower projections]

Despite the advances of Tumor04 over the MatLab™ counterparts, tumors containing greater than 2^{20} cells presented themselves as a memory resource challenge even to personal computers equipped with extra memory. On a Windows NT™ workstation with 128 megabytes (MB) of random access memory (RAM), for example, Tumor04 was successful in generating a tumor with 2^{20} cells, but the demands on the

computer's memory was evident. To test the demands on RAM, Tumor04 was rerun using the following simulation parameters:

- ✓ X_DIM = 400
- ✓ Y_DIM = 400
- ✓ SCALE = 2
- ✓ POP_DOUBLES = 20

These parameters provide a 200^3 growth space and 2^{20} cancer cells. We are able to achieve impressively low simulation times with large tumors because of the use of the growth space cube described in the previous chapter. However, with a growth space of 200^3 , the system requires a minimum of 8,000,000 cells x 32 bits, or approximately 32MB of RAM just for the growth space.

This memory overhead does not include the space required for the cancer cells themselves, nor does it include system resource allocated to the cell vector or any of the GUI components, including the matrix overhead associated with projecting the tumor once growth is complete. Clearly if we are to achieve larger tumor growths, we must make better use of the memory available on even the most modestly configured workstation.

Using the parameters specified above, the system began to slow when the 19th population doubling was approximately 50 percent complete. At this point, the estimated number of cells would be $\frac{3}{2}2^{18}$, or approximately 393,216 cells. The growth to 2^{20} cells was successful, but an error occurred when an attempt was made to show the projection. In general, the amount of memory required for displaying the tumor projection is not significant, but in the case described above, the memory consumption was just enough to remove the last of the remaining system resources.

At 2^{20} cells, the size of the tumor is still far short of the 2^{24} cells required for detection by mammography, and these last four population doublings may be most difficult to achieve given the typical computer RAM constraints outlined above. Nevertheless, the model is extremely fast and the framework may continue to be used for experimentation with other growth models. In order to evaluate the overall speed of the model, 20 simulation runs were completed using 18 population doublings, the default population doublings parameter for Tumor04. The computer environment chosen to run the simulation was a Microsoft Windows NT™ 4.0 workstation with dual Pentium Pro/200 processors and 128MB of RAM. The results of the runs are shown in Figure 5-10.

The mean time for the 18th population doubling was 10.35 seconds, while the cumulative mean for all doublings was 22.83 seconds. Clearly, at nearly 50 percent of the total simulation time, we can see that, at the 18th population doubling, the simulation begins to slow.

Tumor04_1, as discussed earlier, is a model that attempts to simulation the growth of breast cancer tumors without the growth space cube for collision detection. The lack of the cube to test for cell collisions should slow the tumor growth; however, just how much slower the modification would cause a tumor to grow could not be answered with additional time trials.

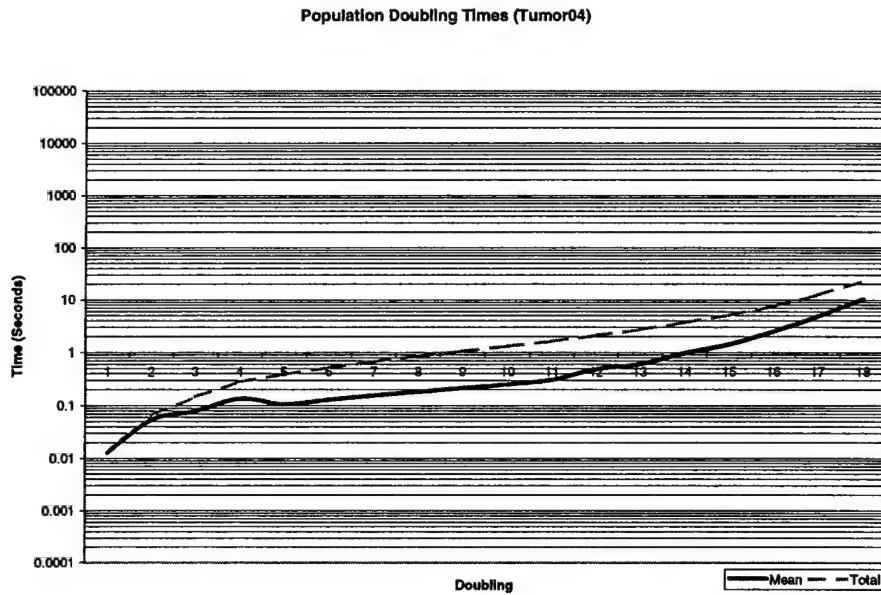


Figure 5-10: Tumor04 simulation results

With the growth space cube removed from the `CancerTumor` class, 20 simulation runs were again completed using Tumor04_1 with 18 population doublings. The computer environment chosen to run the Tumor04_1 simulation was identical to that of Tumor04. The results of the runs are shown in Figure 5-11.

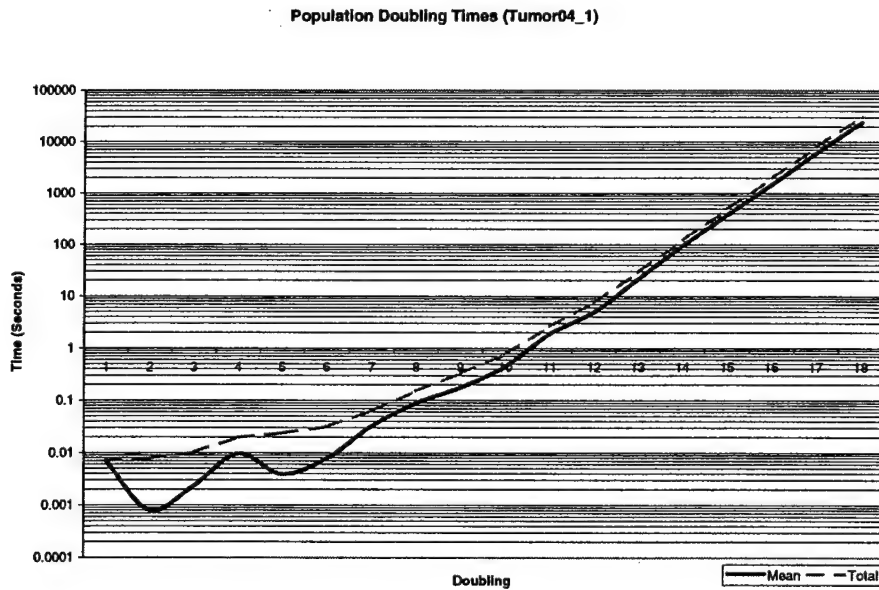


Figure 5-11: Tumor04_1 simulation results

The mean time for the 18th population doubling was 24,074.51 seconds (6.69 hours) while the cumulative mean for all doublings was 32,182.24 seconds (8.94 hours). With this model, we notice a decrease in performance around the 14th population doubling. Clearly, the simulation times for Tumor04_1 are excessive, and dispensing with the growth space cube while other aspects of the architecture remain the same appears to be a poor choice. To support this assertion, we conduct a detailed analysis of the two sets of simulation results. First, the summary data for the comparison are shown in Table 5-1.

Table 5-1: Mean population-doubling times (in seconds)

Doubling	Tumor04	Tumor04_1
1	0.012800	0.00705
2	0.053800	0.00080
3	0.079500	0.00230
4	0.135300	0.00960
5	0.106800	0.00390
6	0.129500	0.00770
7	0.156100	0.03205
8	0.181100	0.08835
9	0.213950	0.17500
10	0.252100	0.44615
11	0.308500	1.88845
12	0.485900	4.97970
13	0.605275	21.33350
14	0.974250	95.32340
15	1.428750	385.22900
16	2.503700	1506.60460
17	4.848250	6091.59230
18	10.352350	24074.51400
Total Time	32.827925	32182.5263

We note from the tabulated data that, without the growth space cube, Tumor04_1 appears to be faster through population doubling nine. After the nine doubling, however, the doubling times for Tumor04_1 grow dramatically, and the cumulative total simulation time for Tumor04_1 is approximately 1,000 times that of Tumor04.

To determine the statistical significance of the data, the differences were taken between the mean run times of each population doubling, and the distribution of the differences was checked for normality using the Shapiro-Wilk Test. At the 95 percent confidence level, virtually all data are *not* normally distributed. This finding precludes the use of traditional methods for determining statistical differences between the means. Instead, we chose non-parametric methods to conduct the analysis. Specifically, the Wilcoxon Signed-Rank Test was conducted using JMP IN® from SAS Institute Inc. Of the 18 population doublings, only the difference in times between the first and ninth doublings proved *insignificant* at the 99 percent confidence level; *all remaining differences are significant*. These findings are summarized in Figure 5-12.

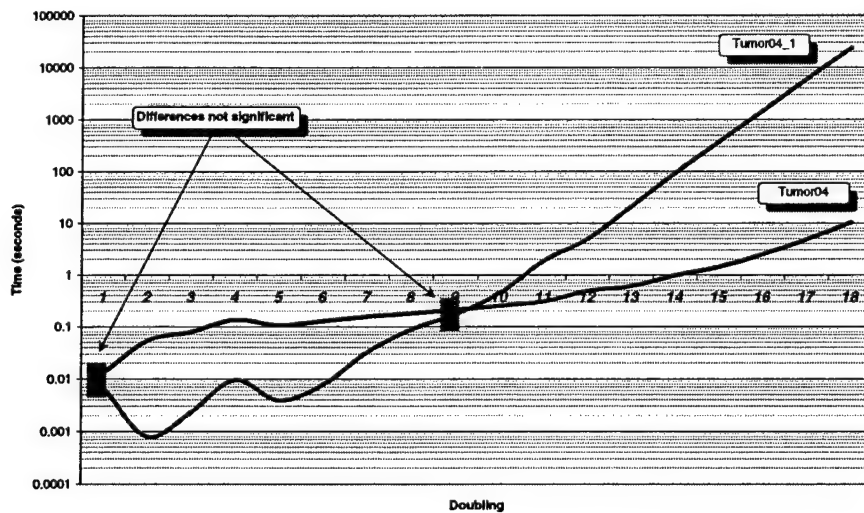


Figure 5-12: Comparison of simulation run times

The final model constructed for this project is Tumor04_2. Based upon the findings above, the tumor growth space cube was returned to the CancerTumor class. Additionally, rather than simply display simulation statistics to the DOS window, these

statistics may be sent to a delimited text file for easy importing into data analysis software. The analysis of statistics such as the elapsed time for population doublings may now be reasonably automated. The final embellishment to Tumor04_2 is the most dramatic in terms of the future potential for modeling cell automaton: the establishment of a rule set for cell behavior. Before we proceed, recall that modifications included in Tumor04_2 result in a context of *simulation time* as opposed to population doublings.

The default simulation parameters for Tumor04_2 are as follows:

- ✓ DEFAULT_X_DIM = 400
- ✓ DEFAULT_Y_DIM = 400
- ✓ DEFAULT_SCALE = 6
- ✓ DEFAULT_TIME = 400

These parameters provide a 66^3 growth space and an unknown number of cancer cells. We wish to conduct a final test that grows the largest tumor possible within a reasonable amount of time, so a single simulation was run using the parameter values of 400, 400, 4, 500, which provides us with a growth space of 100^3 .

The simulation was run without regard to the collection of time data; the `CancerTumor` class was modified to instead capture the number of cancer cells in the tumor at the end of each time period. These data were sent to a text file and then imported into Mathcad® 8 Professional for graphical analysis. The results of the single simulation run are shown in Figure 5-13. Notice the stair step effect that exists during the first half of the simulation.

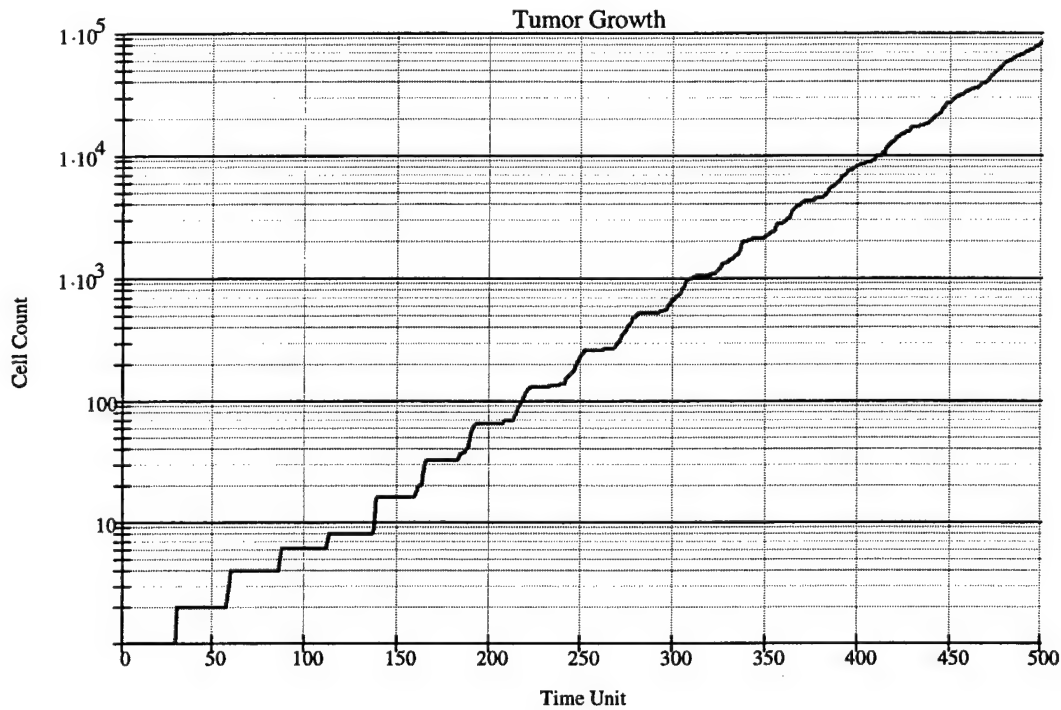


Figure 5-13: Simulation results (Tumor04_2)

A review of the raw data reveals that the first population doubling occurs at $t = 30$. Based upon the rules established for the behavior of the cancer cells, the expected value is as follows:

$$\begin{aligned}
 E[t_{d1}] &= E[\text{Max}(TIP_{G1})] + E[\text{Max}(TIP_S)] + E[\text{Max}(TIP_{G2})] + E[\text{Max}(TIP_M)] \\
 &= 12 + 8 + 7 + 1.5 \\
 &= 28.5 \Rightarrow 29
 \end{aligned}$$

Thus, the observed time of 30 implies that 29 time units were spent in phases G1, S, G2, and M. This corresponds precisely with the expected time. Note that the observed time is consistent with the first stair in Figure 5-13. Close inspection of the second stair step reveals that the number of cells in the tumor increases quickly from one to three. Inspection of the raw data tells us that this occurs at time $t = 58$ (3 cells) and $t = 59$ (4 cells). This indicates that the original cell and the first daughter cell both divide at times

very close to each other. Based upon the process flowchart presented in the previous chapter, this makes sense. Note that as time increases, the time spread between when the cells divide increases until the stair step feature of the graph is lost completely. From the raw data, we see that by $t = 270$ cells are dividing at nearly every time increment. At $t = 500$, the cell count is 86,125. This represents slightly over 16 complete population doublings, where $2^{16} = 64,536$ cells. The simulation completed in approximately 50 seconds.

5.4 Conclusion

Tumor04_2 represents a short evolution and culmination of ideas of the work presented by Bassham and others. The Java platform proved a suitable environment for these ideas, and showed that reasonably large tumors may be grown in 3-D space in a very short amount of time. The object-oriented approach to developing these models should promote their understanding and facilitate additional embellishments.

Clearly, the use of a static 3-dimensional matrix in this environment provides a fast, easy-to-use mechanism for cell collision detection; however, there is some overhead associated with this mechanism that must be taken into consideration. These considerations must be taken seriously as additional attributes are given to individual cancer cells; generally, the increase in "intelligence" of a cancer cell will be directly proportional to the amount of system resources required to maintain each cell. As the model becomes more complex, structures such as the 3-D growth space object must be reconsidered. What follows are some ideas on taking the next step toward fully realizing a simulated tumor that is detectable using current mammography techniques.

5.5 Recommendations

There are three recommendations for future embellishments that may pave the way for growing and visualizing tumor sizes of 2^{24} cells or greater. These recommendations fall in the area system resource consumption, the types of data structures used to manage the cancer cells, and then ultimately visualizing the tumor at the completion of the simulation.

The analysis completed here shows that performance problems exist for large tumors. Some of the performance problems, specifically memory allocation and recovery, may lie in *when* and *how* objects are created and then disposed of. Dennis Sosnoski, a senior software consultant based in Washington, has available an on-line paper entitled, "Java performance programming, Part 1: Smart object management saves the day." This paper provides some performance comparisons using different Java object types. For example, the paper explains the implications of using the `String` class to represent a "1" verses using the `Integer` class. While this distinction is trivial for small tumors, at 2^{24} cells it could be critical.

This recommendation is made after trivial modifications were made to the model that showed the system resource level after each simulation time increment. As the tumor grew larger, the system resources were consumed much more quickly. Most often, just before the point of complete consumption of all memory resources, there was a pause in the simulation while the Java garbage collection (memory reclamation) routine went to work. If the number and size of objects created can be limited, the memory consumption can be reduced. Additionally, it may be possible to initiate garbage collection at points critical to the simulation but not obvious to the garbage collection mechanism.

Sosnoski's paper is readable even for the novice programmer and may provide some insight into why the current model experiences memory resource shortages at 2^{20} population doublings, which currently is short of our objective. This paper is available at the following Web URL: <http://www.javaworld.com/javaworld/jw-11-1999/jw-11-performance.html>.

The second recommendation is related to the type of objects used to manage the cancer tumor and cells. There are two components to the current model that may be eliminated entirely, and the net effect could be a savings in both time and system resources.

The only purpose of the 3-D growth space is to provide nearly instantaneous confirmation that a candidate cell position is already occupied. As we proved in Tumor04_1, removing this structure merely impacts the simulation time; it has no effect on the development of the tumor itself. In Tumor04_1 we performed continuous searches of the cell vector object to obtain the same collision information. As we also proved, however, the time required to perform these checks is unacceptable. The *Java Collections Framework* may offer a solution.

Tumor04_2 introduces the Collections Framework by implementing an instance of the `ArrayList` class. This class, as well as the `Vector` class, provides the means to easily manipulate virtually any type of object contained in the list. Included are methods for searching lists for specific data items. Unfortunately, these classes are part of the `Java List` interface, and do not automatically prevent duplicate objects—two cancer cells with the same coordinates, for example. The Collections Framework consists of two other interfaces that may provide a solution: The `Set` interface and the `Map` interface. The

Set interface is implemented with a HashSet or TreeSet, while the Map interface is implemented with a HashMap or TreeMap. These classes offer searching and sorting methods not available in the List interface. It may be possible to use each xyz tuple as a unique “key” for nearly instantaneous access to the pool of cells, thereby achieving the collision detection required for the algorithm. It is advisable that whoever chooses to investigate the Collections Framework to review an introductory paper by the MageLang Institute. The Collections Framework paper is available on-line at the following URL: <http://developer.java.sun.com/developer/onlineTraining/collections/index.html>.

The final recommendation is not directly related to performance; rather, it is tied to visualization of the tumor. Visualization can be a thesis in itself and details are not provided here. In a nutshell, the Java3D API, from which we already implement the Point3f class, offers a multitude of multimedia extensions. These extensions include sophisticated methods for rendering complex 3-dimensional data. One of the recommendations made by Bassham in his thesis was related to potential surface rendering of the tumor. Java3D offers methods for object collisions, geometry, visual data compression, and multiple views. This just scratches the surface of what the Java3D API might offer. Specific information regarding the user of Java3D may be found online at the following URL: <http://java.sun.com/products/java-media/3D/collateral/>

Appendix A: Java Source Code - Tumor04

Package Name	Class Name	Source Filename
Tumor04	CancerSimApp	CancerSimApp.java

```

1:  /*
2:  Title.....: CancerSim
3:  Version.....: 4 (06 Dec 1999)
4:  Copyright...: Copyright (c) 1999
5:  Author.....: Capt Bruce C Jenkins (GOA-00M)
6:  Company.....: Air Force Institute of Technology
7:  Description: CancerSim simulates the growth of a 3-dimensional breast cancer
8:  tumor. This model utilizes a static 1 x m x n cube for the tumor growth space.
9:  The dimensions of the growth space depend on the pixel dimensions chosen for the
10: applet and the scale (size) of the cancer cell in pixels. After initialization,
11: a single cancer cell is positioned approximately in the center of the cube. The
12: cell's position is tracked with a dynamic, object-oriented container (Vector)
13: that is created at runtime. The number of times the container is examined
14: represents the number of population doublings that occur. For each cancer cell
15: found in the vector ("cellVector"), a new cancer cell is placed in an unoccupied
16: position in the cube ("growthSpace"). The direction in which a search for a
17: free space is conducted (there are 26 possible directions) is determined by a
18: uniform random number, which is generated each time a cell must be positioned.
19: (See the written thesis for additional documentation.) The simulation ends when
20: the number of population doublings is complete or the tumor growth has exceeded
21: the bounds of the growthSpace.
22:
23: This version of CancerSim includes the following enhancements:
24:
25: 1 In the previous version, cellWidth x cellWidth color objects were created
26: each time the tumor was painted to the screen. There was a significant amount
27: of memory and time overhead associated with this method. In this version of
28: CancerSim, the number of color objects required is reduced by
29: cellWidth*cellWidth. After the number of cells (of the face of the cube) are
30: determined, a color chart is created that contains the range of color shades
31: scaled to the maximum possible number of cells along any of the three axis.
32:
33: 2. The above code was moved to a new class: ControlPanel. This modification is
34: part of a complete restructuring of the program to take advantage of the Java
35: Swing componenets.
36:
37: 3. The applet was converted to an application for the same reason identified
38: above.
39:
40: */
41: package Tumor04;
42:
43: import javax.swing.UIManager;
44:
45: public class CancerSimApp {
46:
47:     /******* Simulation Defaults *****/
48:     private final int DEFAULT_X_DIM = 400; // Width and height of of tumor
49:     private final int DEFAULT_Y_DIM = 400; // display window.
50:     private final int DEFAULT_POP_DOUBLES = 18; // Population doublings.
51:     private final int DEFAULT_SCALE = 4; // Display cell size.
52:     /*******
53:
54:     static ControlPanel controlPanel; // Simulation control component.
55:     boolean packFrame = false; // Default packing of frames.
56:
57:     // Construct the application.
58:     // Inputs: None
59:     // Return: n/a
60:     public CancerSimApp() {
61:         controlPanel = new ControlPanel(); // Create a new control panel object.
62:         // Pack frames that have useful preferred size info. For example, from

```

```

63:         // their layout.
64:         if (packFrame)
65:             controlPanel.pack();
66:         else
67:             controlPanel.validate(); // Validate frames that have preset sizes.
68:         controlPanel.setVisible(true);
69:         // Start control panel with defaults.
70:         controlPanel.run(DEFAULT_X_DIM, DEFAULT_Y_DIM,
71:             DEFAULT_POP_DOUBLES, DEFAULT_SCALE);
72:     }
73:
74:     // Main method. Establish the application's look and feel and instantiate
75:     // the application. For consistency from platform to platform, the look
76:     // and feel is set to "Metal."
77:     // Inputs: None
78:     // Return: n/a
79:     public static void main(String[] args) {
80:         try {
81:             UIManager.setLookAndFeel("javax.swing.plaf.motif.MetalLookAndFeel");
82:         }
83:         catch(Exception e) {
84:         }
85:         CancerSimApp cancerSim = new CancerSimApp();
86:     }
87: }

```

Package Name	Class Name	Source Filename
Tumor04	CancerCell	CancerCell.java

```

1: //Title.....: CancerCell
2: //Version.....: 3.1 (07 Dec 1999)
3: //Copyright...: Copyright (c) 1999
4: //Author.....: Capt Bruce C Jenkins (GOA-00M)
5: //Company.....: Air Force Institute of Technology
6: /*Description: CancerCell is a key components of this simulation. It is
7: intended that this class have the capability to scale with the increased
8: complexity inherent in improving the CancerSim application.
9:
10: This version of CancerCell includes the following enhancements:
11:
12: - The Point class was replaced with Point3f from Java3D. This was done to
13: allow complex vector calculations between two points in 3D space.
14: */
15:
16: package Tumor04;
17:
18: import javax.vecmath.*;
19:
20: public class CancerCell {
21:
22:     Point3f    location;        // The xyz coordinates of the cancer cell.
23:     boolean    isAlive = true;  // Indicates if cell is alive or dead.
24:     boolean    canDivide;       // Indicates if cell can be divided.
25:
26:     // Create the constructor so that coordinates are required at instantiation.
27:     // Inputs: Coordinates for new cancer cell
28:     // Return: n/a
29:     CancerCell(float xPos, float yPos, float zPos) {
30:         this.init(xPos, yPos, zPos);
31:     }
32:
33:     // Create constructor so that no coordinates are required at instantiation.
34:     // Inputs: None
35:     // Return: n/a
36:     CancerCell() {
37:     }
38:

```

```

39: // Set the current location of the cancer cell.
40: // Inputs: Coordinates for new cancer cell
41: // Return: n/a
42: private void setLocation(float xPos, float yPos, float zPos) {
43:     location.set(xPos, yPos, zPos);
44: }
45:
46: private void init(float x, float y, float z) {
47:     location = new Point3f(x, y, z);
48: }
49:
50: // Get the current location of the cancer cell.
51: // Inputs: None
52: // Return: Location of cancer cell
53: private Point3f getLocation() {
54:     return this.location;
55: }
56:
57: // Set the divide status of the cancer cell.
58: // Inputs: Boolean value indicating if cell is allowed to divide
59: // Return: n/a
60: private void setCanDivide(boolean allowed) {
61:     this.canDivide = allowed;
62: }
63:
64: // Get the divide status of the cancer cell.
65: // Inputs: None
66: // Return: Boolean value indicating if cell is allowed to divide
67: private boolean getCanDivide() {
68:     return this.canDivide;
69: }
70:
71: // Set the dead/alive condition of the cancer cell. If the cell is alive,
72: // the ability to divide is assumed.
73: // Inputs: Boolean value indicating if cell is alive or dead
74: // Return: n/a
75: private void setIsAlive(boolean condition) {
76:     this.isAlive = condition;
77:     if (this.isAlive) {
78:         this.setCanDivide(true);
79:     }
80:     else
81:         this.setCanDivide(false);
82: }
83:
84: // Get the dead/alive condition of the cancer cell.
85: // Inputs: None
86: // Return: Boolean value indicating if cell is alive or dead
87: private boolean getIsAlive(){
88:     return this.isAlive;
89: }
90:
91: // Start the process of cell division. While in the state of dividing, a new
92: // cancer cell is created, it (1) determine a direction of travel, and
93: // (2) finds a free space into which it can place itself.
94: // Inputs: None
95: // Return: n/a
96: private void divide() {
97:     if (this.canDivide) {
98:         float x = this.location.x;
99:         float y = this.location.y;
100:         float z = this.location.z;
101:         new CancerCell(x, y, z);
102:     }
103: }
104: }

```

Package Name	Class Name	Source Filename
Tumor04	CellDirection	CellDirection.java

```

1: //Title.....: CellDirection
2: //Version.....: 2.0 (07 Nov 1999)
3: //Copyright...: Copyright (c) 1999
4: //Author.....: Capt Bruce C Jenkins (GOA-00M)
5: //Company.....: Air Force Institute of Technology
6: /*Description: This class contains the methods necessary for a cancer cell to
7: find a free space within the tumor. A static array is used to divide the number
8: of possible directions of travel within the cube into evenly distributed cells.
9: A uniform random number is generated and a cell into which the random number
10: "fits" is found. The corresponding xyz tuple is then pulled and passed back to
11: the calling module. The tuple represents the planned direction of travel for
12: the cancer cell.
13:
14: Modification history:
15:
16: 1. The user-written class Point was replaced with Point3f from the Java3D
17: library. Point 3f includes methods for manipulating vectors in 3-D space.
18: */
19: package Tumor04_2;
20:
21: import java.lang.*;
22: import java.lang.Math.*;
23: import javax.vecmath.*;
24:
25: class CellDirection {
26:
27:     static final int rRef = 0;
28:     static final int X   = 1;
29:     static final int Y   = 2;
30:     static final int Z   = 3;
31:     static Point3f tPoint = new Point3f(0, 0, 0);
32:     static final double rDirection[][] =
33:         {{0.038462, 0, 0, -1}, // "-Z" ***** [ 0]
34:          {0.076923, 0, 1, -1}, // N [ 1]
35:          {0.115385, 1, 1, -1}, // NE [ 2]
36:          {0.153846, 1, 0, -1}, // E [ 3]
37:          {0.192308, 1, -1, -1}, // SE [ 4]
38:          {0.230769, 0, -1, -1}, // S [ 5]
39:          {0.269231, -1, -1, -1}, // SW [ 6]
40:          {0.307692, -1, 0, -1}, // W [ 7]
41:          {0.346154, -1, 1, -1}, // NW [ 8]
42:          {0.384615, 0, 1, 0}, // N "Z" ***** [ 9]
43:          {0.423077, 1, 1, 0}, // NE [10]
44:          {0.461538, 1, 0, 0}, // E [11]
45:          {0.500000, 1, -1, 0}, // SE [12]
46:          {0.538462, 0, -1, 0}, // S [13]
47:          {0.576923, -1, -1, 0}, // SW [14]
48:          {0.615385, -1, 0, 0}, // W [15]
49:          {0.653846, -1, 1, 0}, // NW [16]
50:          {0.692308, 0, 0, 1}, // "+Z" ***** [17]
51:          {0.730769, 0, 1, 1}, // N [18]
52:          {0.769231, 1, 1, 1}, // NE [19]
53:          {0.807692, 1, 0, 1}, // E [20]
54:          {0.846154, 1, -1, 1}, // SE [21]
55:          {0.884615, 0, -1, 1}, // S [22]
56:          {0.923077, -1, -1, 1}, // SW [23]
57:          {0.961538, -1, 0, 1}, // W [24]
58:          {1.000000, -1, 1, 1}}; // NW [25]
59:
60:     CellDirection() {} // Constructor.
61:
62:     // Generate random number, look up direction tuple, and return a point
63:     // containing the tuple.
64:     // Inputs: None
65:     // Return: Point
66:     Point3f getDirection() {

```

```

67:
68:     int i = 0;
69:     double rNum = Math.random(); // Uniform, on the interval [0.000, 1.000).
70:     // Search through the list of directions and find a match for the random
71:     // number drawn.
72:     while (rNum > rDirection[i][rRef]) {
73:         i++;
74:     }
75:     tPoint.x = (float)rDirection[i][X];
76:     tPoint.y = (float)rDirection[i][Y];
77:     tPoint.z = (float)rDirection[i][Z];
78:     return tPoint;
79: }
80: }

```

Package Name	Class Name	Source Filename
Tumor04	CancerTumor	CancerTumor.java

```

1: //Title.....: CancerTumor
2: //Version.....: 2.1 (09 Jan 2000)
3: //Copyright...: Copyright (c) 1999
4: //Author.....: Capt Bruce C Jenkins (GOA-00M)
5: //Company.....: Air Force Institute of Technology
6: /*Description: This class is implemented as the container for all cancer cells
7: created in this simulation. It is intended to be the "brains" of the collection
8: of cells. It will keep track of the number of cells that exist, control when
9: and how many times a cell may divide, their rate of mitosis (death), as well as
10: other factors influencing the behavior of the cells as a collective entity.
11:
12: This initial version of CancerTumor replaces the "cellVector" in CancerSim v2.
13: Previously, the intent was to create a CancerTumor "containing" a cellVector
14: object. After additional research, it makes better sense to simply code this
15: class so that CancerTumor, in essence, "is" the cellVector. Because of as yet
16: unforeseen embellishments, this object will be designed to run in its own thread.
17: If it turns out that a separate thread is not required, this feature is easily
18: removed from the object.
19: */
20:
21: package Tumor04;
22:
23: import java.util.*;
24: import java.awt.*;
25: import javax.vecmath.*;
26: import java.awt.Toolkit;
27:
28: class CancerTumor extends Thread {
29:
30:
31:     private int    cellCount = 0;        // Number of cells in the cancer tumor.
32:     private boolean stopTumor = false;   // Flag to stop tumor growth.
33:     private int    popDoubles;           // Number of times cell pop may divide.
34:     private int    cellSize;             // Cell size (resolution) in pixels.
35:     // changed byte ==> int 13 Feb 2000 to correct projection display errors.
36:     private int    growthSpace [][][];   // Tumor growth space (cube).
37:     private int    cubeWidth, cubeHeight, cubeDepth;
38:     private int    xPos, yPos, zPos;
39:     private CellDirection cellDirection;
40:     private Point3f tPoint, tumorOrigin;
41:     private Vector  tumor;
42:     private float   cellDistance,
43:                   sumDistance,
44:                   meanDistance,
45:                   maxDistance = 0;
46:     Date currentTime;
47:     long startTime, stopTime, elapsedTime;
48:
49:

```

```

50:    // Construct the tumor by creating a new cancer cell at postion x, y, z.
51:    // Inputs: Coordinates of new cancer cell
52:    // Return: n/a
53:    CancerTumor (int popDoubles,
54:                int cellSize,
55:                int cellWidth,
56:                int cellHeight,
57:                int cellDepth,
58:                int FirstCellX,
59:                int FirstCellY,
60:                int FirstCellZ) {
61:        // Set initial tumor capacity to 512; doubles when needed.
62:        tumor = new Vector(512);
63:        this.popDoubles = popDoubles;
64:        this.cellSize = cellSize;
65:        this.cubeWidth = cellWidth;
66:        this.cubeHeight = cellHeight;
67:        this.cubeDepth = cellDepth;
68:        this.xPos = FirstCellX;
69:        this.yPos = FirstCellY;
70:        this.zPos = FirstCellZ;
71:    }
72:
73:
74:    // Initialize the growth space cube to zeros, place first cell and update
75:    // all three facings.
76:    // Inputs: None
77:    // Return: n/a
78:    private void init() {
79:        this.setStop(false);
80:        // Create the cube.
81:        this.growthSpace = new int [cubeDepth][cubeHeight][cubeDepth];
82:        // Initialize the cube.
83:        for (int i = 0; i < cubeWidth; i++) {
84:            for (int j = 0; j < cubeHeight; j++) {
85:                for (int k = 0; k < cubeDepth; k++) {
86:                    growthSpace[i][j][k] = 0; // Set each cell to "white."
87:                }
88:            }
89:        }
90:        // Place first cancer cell in growth space.
91:        growthSpace[xPos][yPos][zPos] = 1;
92:        // Update faces of the cube.
93:        growthSpace[xPos][yPos][0] = 1;
94:        growthSpace[xPos][0][zPos] = 1;
95:        growthSpace[0][yPos][zPos] = 1;
96:        // Place the first cancer cell in the tumor.
97:        tumor.addElement(new CancerCell(xPos, yPos, zPos));
98:        // ID the origin of the cancer tumor.
99:        this.tumorOrigin = new Point3f(xPos, yPos, zPos);
100:        // Instantiate the "direction finder."
101:        this.cellDirection = new CellDirection();
102:    }
103:
104:    // Set the cell count to the number of cells currently in the tumor.
105:    // Inputs: None
106:    // Return: n/a
107:    public void setCellCount() {
108:        this.cellCount = tumor.size();
109:    }
110:
111:    // Get the number of cells currently in the tumor.
112:    // Inputs: None
113:    // Return: The number of cells currently in the tumor
114:    public int getCellCount() {
115:        return this.cellCount;
116:    }
117:
118:    // Set the stop flag.
119:    // Inputs: Condition of flag: set (true) or not (false)
120:    // Return: n/a
121:    private void setStop(boolean condition) {
122:        this.stopTumor = condition;
123:    }

```

```

124:
125: // Get the stop flag.
126: // Inputs: None
127: // Return: Condition of flag: set (true) or not (false)
128: public boolean getStop() {
129:     return this.stopTumor;
130: }
131:
132: // Get a copy of the cube representing the X projection.
133: // Inputs: None
134: // Return: Projection Matrix
135: public int[][] getFaceX() {
136:     int [][] face = new int[cubeWidth][cubeDepth];
137:     for (int y = 0; y < this.cubeHeight; y++) {
138:         for (int z = 0; z < this.cubeDepth; z++) {
139:             face[y][z] = this.growthSpace[0][y][z];
140:         }
141:     }
142:     return face;
143: }
144:
145: // Get a copy of the cube representing the Y projection.
146: // Inputs: None
147: // Return: Projection Matrix
148: public int[][] getFaceY() {
149:     int [][] face = new int[cubeHeight][cubeDepth];
150:     for (int x = 0; x < this.cubeWidth; x++) {
151:         for (int z = 0; z < this.cubeDepth; z++) {
152:             face[x][z] = this.growthSpace[x][0][z];
153:         }
154:     }
155:     return face;
156: }
157:
158: // Get a copy of the cube representing the Z projection.
159: // Inputs: None
160: // Return: Projection Matrix
161: public int[][] getFaceZ() {
162:     int [][] face = new int[cubeWidth][cubeHeight];
163:     for (int x = 0; x < this.cubeWidth; x++) {
164:         for (int y = 0; y < this.cubeHeight; y++) {
165:             face[x][y] = this.growthSpace[x][y][0];
166:         }
167:     }
168:     return face;
169: }
170:
171:
172:
173: // This method will be used to execute the tumor on its own thread.
174: // Inputs: None
175: // Return: n/a
176: public void run() {
177:     init();
178:     // For each population doubling, replicate all current cancer cells. New
179:     // daughter cells are set off with a marker so that they are not replicated
180:     // until the next doubling.
181:     int firstCell = 0;
182:     int lastCell = firstCell;
183:     int lastDaughter = lastCell;
184:     int moveX, moveY, moveZ;
185:     CancerCell tCell = new CancerCell(); // Temporary cell.
186:     CancerSimApp.controlPanel.jProgressBar_cellDoubles.setStringPainted(true);
187:     currentTime = new Date();
188:     startTime = currentTime.getTime();
189:     for (int i = 1; i <= popDoubles; i++) {
190:         CancerSimApp.controlPanel.jProgressBar_popDoubles.setValue(i);
191:         CancerSimApp.controlPanel.jProgressBar_popDoubles.setString("Doubling "
192:             + Integer.toString(i));
193:         // Cycle through the cell vector and replicate each cell.
194:         CancerSimApp.controlPanel.jProgressBar_cellDoubles.setMaximum(lastCell);
195:         for (int j = firstCell; j <= lastCell; j++) {
196:             if (j%100 == 0) {
197:                 CancerSimApp.controlPanel.jProgressBar_cellDoubles.setValue(j);

```



```

198:         }
199:         // Determine direction of new cell.
200:         tPoint = cellDirection.getDirection();
201:         moveX = (int)tPoint.x;
202:         moveY = (int)tPoint.y;
203:         moveZ = (int)tPoint.z;
204:         // Access current cancer cell and retrieve coordinates.
205:         tCell = (CancerCell)tumor.elementAt(j);
206:         xPos = (int)tCell.location.x;
207:         yPos = (int)tCell.location.y;
208:         zPos = (int)tCell.location.z;
209:         // Enter the following loop if the growth space is not empty (i.e.,
210:         // it's occupied by a cancer cell) at the current coordinates and the
211:         // offsets (directions of travel). Within the loop, increment the
212:         // offsets by 1 if they are positive, by -1 if negative. If the
213:         // offset is 0, no action is taken.
214:         while (growthSpace[xPos + moveX][yPos + moveY][zPos + moveZ] != 0) {
215:             if (moveX > 0) moveX++; // Continue in positive X direction.
216:             else if (moveX < 0) moveX--; // Continue in negative X direction.
217:             if (moveY > 0) moveY++; // Continue in positive Y direction.
218:             else if (moveY < 0) moveY--; // Continue in negative Y direction.
219:             if (moveZ > 0) moveZ++; // Continue in positive Z direction.
220:             else if (moveZ < 0) moveZ--; // Continue in negative Z direction.
221:         } // end while
222:         xPos += moveX; // ==> xPos = xPos + moveX
223:         yPos += moveY; // ==> yPos = yPos + moveY
224:         zPos += moveZ; // ==> zPos = zPos + moveZ
225:         // Place the cancer cell in the growth space.
226:         growthSpace[xPos][yPos][zPos] = 1; // Red cancer cell.
227:         // Update the "facings."
228:         growthSpace[xPos][yPos][0] += 1;
229:         growthSpace[xPos][0][zPos] += 1;
230:         growthSpace[0][yPos][zPos] += 1;
231:         // Update the index to the last daughter cell and add the daughter
232:         // cell to the tumor.
233:         lastDaughter++;
234:         tumor.addElement(new CancerCell(xPos, yPos, zPos));
235:         cellCount++;
236:         // Access newly placed cell and calculate distance from origin.
237:         tCell = (CancerCell)tumor.elementAt(lastDaughter);
238:         cellDistance = tCell.location.distance(tumorOrigin);
239:         if (cellDistance > maxDistance) maxDistance = cellDistance;
240:         sumDistance += cellDistance;
241:     }
242:     // A population doubling is complete. The added daughter cells are now
243:     // part of the set of cancer cells that are candidates for the next
244:     // population doubling. Update the index of cancer cells to include the
245:     // new daughter cells.
246:     lastCell = lastDaughter;
247:     currentTime = new Date();
248:     stopTime = currentTime.getTime();
249:     elapsedTime = stopTime - startTime;
250:     System.out.println("Doubling " + i + " elapsed time: " +
251:         elapsedTime/1000.000);
252: } // end for
253: CancerSimApp.controlPanel.jProgressBar_cellDoubles.setValue(lastCell);
254: CancerSimApp.controlPanel.jProgressBar_popDoubles.setString("Done!");
255: Toolkit.getDefaultToolkit().beep();
256: CancerSimApp.controlPanel.stopButton.setEnabled(false);
257: CancerSimApp.controlPanel.startButton.setEnabled(true);
258: System.out.println("Maximum distance from tumor origin: " +
259:     maxDistance + " units.");
260: System.out.println("Mean distance from tumor origin...: " +
261:     sumDistance / cellCount + " units.");
262: this.setStop(true);
263: } // end run
264:
265: }

```

Package Name	Class Name	Source Filename
Tumor04	ControlPanel	ControlPanel.java

```

1:  /*
2:  Title.....: ControlPanel
3:  Version.....: 1.0 (10 Dec 1999)
4:  Copyright...: Copyright (c) 1999
5:  Author.....: Capt Bruce C Jenkins (GOA-00M)
6:  Company.....: Air Force Institute of Technology
7:  Description: This class is designed to be the main point of control for the
8:  cancer simulation. It deviates from previously developed classes in that it
9:  makes use of Java Swing user interface (UI) componenets. The control panel
10: receives the simulation defaults from the main application and uses them to
11: initiate tumor growth.
12: */
13:
14: package Tumor04;
15:
16: import java.awt.*;
17: import java.awt.event.*;
18: import javax.swing.*;
19: import javax.swing.border.*;
20: import javax.swing.event.*;
21:
22:
23:
24: public class ControlPanel extends JFrame {
25:
26:     private int popDoubles; // Number of population doublings.
27:     private int cellSize;   // Display cell size.
28:     private int cellWidth, cellHeight, cellDepth; // Cube dimensions.
29:     private int windowWidth, windowHeight; // Window dimensions.
30:     private int firstCellX, firstCellY, firstCellZ; // Tumor origin.
31:
32:     private CancerTumor cancerTumor;
33:     private TumorWindow tumorWindow;
34:
35:     // The objects below are the primary components of the control panel--the
36:     // applications's graphical user interface.
37:     JPanel jPanel1 = new JPanel();
38:     JButton startButton = new JButton();
39:     JButton stopButton = new JButton();
40:     JPanel jPanel2 = new JPanel();
41:     JPanel jPanel3 = new JPanel();
42:     JTextField jTextField_popDoubles = new JTextField();
43:     JLabel jLabel1 = new JLabel();
44:     TitledBorder titledBorder1;
45:     TitledBorder titledBorder2;
46:     TitledBorder titledBorder3;
47:     JLabel jLabel2 = new JLabel();
48:     JTextField jTextField_cellResolution = new JTextField();
49:     JCheckBox jCheckBox_ShowTumorWindow = new JCheckBox();
50:     BorderLayout borderLayout1 = new BorderLayout();
51:     JPanel jPanel4 = new JPanel();
52:     JRadioButton jRadioButton_xAxis = new JRadioButton();
53:     JRadioButton jRadioButton_yAxis = new JRadioButton();
54:     JRadioButton jRadioButton_zAxis = new JRadioButton();
55:     JPanel jPanel6 = new JPanel();
56:     JProgressBar jProgressBar_popDoubles = new JProgressBar();
57:     JProgressBar jProgressBar_cellDoubles = new JProgressBar();
58:     JPopupMenu jPopupMenu1 = new JPopupMenu();
59:     JLabel jLabel3 = new JLabel();
60:     JTextField jTextField_windowWidth = new JTextField();
61:     JTextField jTextField_windowHeight = new JTextField();
62:     JLabel jLabel4 = new JLabel();
63:     JLabel jLabel5 = new JLabel();
64:     GridBagLayout gridBagLayout3 = new GridBagLayout();
65:     GridBagLayout gridBagLayout4 = new GridBagLayout();
66:     GridBagLayout gridBagLayout5 = new GridBagLayout();

```

```

67: GridBagLayout gridBagLayout2 = new GridBagLayout();
68: GridBagLayout gridBagLayout1 = new GridBagLayout();
69:
70:
71:
72:
73: // Construct the frame
74: public ControlPanel() {
75:     enableEvents(AWTEvent.WINDOW_EVENT_MASK);
76:     try {
77:         // A visual components are in this method so that JBuilder knows where
78:         // to look to build the UI. This method does not preclude the use of
79:         // alternative RAD tools or the command line Java compiler.
80:         jbInit();
81:     }
82:     catch(Exception e) {
83:         e.printStackTrace();
84:     }
85: }
86:
87: // Component initialization. The method jbInit is required by visual
88: // designer subsystem of Borland's JBuilder (by Inprise Corporation).
89: private void jbInit() throws Exception {
90:     titledBorder1 = new TitledBorder("Parameters");
91:     titledBorder2 = new TitledBorder("Projection");
92:     titledBorder3 = new TitledBorder("Progress");
93:     setSize(new Dimension(529, 154));
94:     setTitle("CancerSim");
95:     setResizable(false);
96:     this.getContentPane().setLayout(borderLayout1);
97:     jPanel1.setLayout(gridBagLayout2);
98:     startButton.setText("Start");
99:     startButton.addActionListener(new java.awt.event.ActionListener() {
100:         public void actionPerformed(ActionEvent e) {
101:             startButton_actionPerformed(e);
102:         }
103:     });
104:     stopButton.setText("Stop");
105:     stopButton.addActionListener(new java.awt.event.ActionListener() {
106:         public void actionPerformed(ActionEvent e) {
107:             stopButton_actionPerformed(e);
108:         }
109:     });
110:     jPanel2.setLayout(gridBagLayout3);
111:     jPanel3.setLayout(gridBagLayout4);
112:     jTextField_popDoubles.setText("");
113:     jTextField_popDoubles.setHorizontalAlignment(SwingConstants.RIGHT);
114:     jLabel1.setFont(new java.awt.Font("SansSerif", 0, 12));
115:     jLabel1.setHorizontalAlignment(SwingConstants.RIGHT);
116:     jLabel1.setText("Population Doubles:");
117:     jPanel3.setBorder(titledBorder1);
118:     jLabel2.setText("Cell Resolution:");
119:     jLabel2.setFont(new java.awt.Font("SansSerif", 0, 12));
120:     jLabel2.setHorizontalAlignment(SwingConstants.RIGHT);
121:     jTextField_cellResolution.setText("");
122:     jTextField_cellResolution.setHorizontalAlignment(SwingConstants.RIGHT);
123:     jCheckBox_ShowTumorWindow.setText("Show Tumor");
124:     jCheckBox_ShowTumorWindow.addActionListener(new java.awt.event.ActionListen
125:         public void actionPerformed(ActionEvent e) {
126:             jCheckBox_ShowTumorWindow_actionPerformed(e);
127:         }
128:     });
129:     jPanel4.setLayout(gridBagLayout5);
130:     jPanel4.setBorder(titledBorder2);
131:     jRadioButton_xAxis.setText("X");
132:     jRadioButton_xAxis.setSelected(true);
133:     jRadioButton_xAxis.setEnabled(false);
134:     jRadioButton_yAxis.setEnabled(false);
135:     jRadioButton_zAxis.setEnabled(false);
136:     jRadioButton_xAxis.addActionListener(new java.awt.event.ActionListener() {
137:         public void actionPerformed(ActionEvent e) {
138:             jRadioButton_xAxis_actionPerformed(e);
139:         }
140:     });

```

```

141:     jButton_yAxis.setText("Y");
142:     jButton_yAxis.addActionListener(new java.awt.event.ActionListener() {
143:         public void actionPerformed(ActionEvent e) {
144:             jButton_yAxis_actionPerformed(e);
145:         }
146:     });
147:     jButton_zAxis.setText("Z");
148:     jButton_zAxis.addActionListener(new java.awt.event.ActionListener() {
149:         public void actionPerformed(ActionEvent e) {
150:             jButton_zAxis_actionPerformed(e);
151:         }
152:     });
153:     jPanel6.setLayout(gridBagLayout1);
154:     jPanel6.setBorder(titledBorder3);
155:     jLabel3.setHorizontalAlignment(SwingConstants.RIGHT);
156:     jLabel3.setFont(new java.awt.Font("SansSerif", 0, 12));
157:     jLabel3.setText("Window Resolution:");
158:     jTextField_windowWidth.setText("");
159:     jTextField_windowWidth.setHorizontalAlignment(SwingConstants.RIGHT);
160:     jTextField_windowHeight.setText("");
161:     jTextField_windowHeight.setHorizontalAlignment(SwingConstants.RIGHT);
162:     jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
163:     jLabel4.setText("X");
164:     jLabel5.setText("Y");
165:     jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
166:     this.getContentPane().add(jPanel1, BorderLayout.CENTER);
167:     jPanel1.add(jPanel3, new GridBagConstraints(1, 0, 1, 2, 1.0, 1.0
168:         ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
169:         new Insets(0, 0, 3, 0), -11, 1));
170:     jPanel3.add(jTextField_popDoubles,
171:         new GridBagConstraints(3, 0, 1, 1, 1.0, 0.0
172:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
173:         new Insets(-2, 0, 0, 3), 24, 0));
174:     jPanel3.add(jLabel1, new GridBagConstraints(0, 0, 2, 1, 0.0, 0.0
175:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
176:         new Insets(-2, 16, 0, 0), 6, 0));
177:     jPanel3.add(jTextField_windowHeight,
178:         new GridBagConstraints(3, 3, 1, 1, 1.0, 0.0
179:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
180:         new Insets(0, 0, 3, 3), 24, 0));
181:     jPanel3.add(jLabel3, new GridBagConstraints(0, 2, 2, 1, 0.0, 0.0
182:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
183:         new Insets(9, 6, 0, 0), 18, 0));
184:     jPanel3.add(jTextField_windowWidth,
185:         new GridBagConstraints(1, 3, 1, 1, 1.0, 0.0
186:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
187:         new Insets(0, 0, 3, 15), 24, 0));
188:     jPanel3.add(jTextField_cellResolution,
189:         new GridBagConstraints(3, 1, 1, 1, 1.0, 0.0
190:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
191:         new Insets(7, 0, 0, 3), 24, 0));
192:     jPanel3.add(jLabel2, new GridBagConstraints(0, 1, 2, 1, 0.0, 0.0
193:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
194:         new Insets(9, 16, 0, 0), 30, 0));
195:     jPanel3.add(jLabel4, new GridBagConstraints(0, 3, 1, 1, 0.0, 0.0
196:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
197:         new Insets(0, 72, 3, 0), 7, 0));
198:     jPanel3.add(jLabel5, new GridBagConstraints(2, 3, 1, 1, 0.0, 0.0
199:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
200:         new Insets(0, 0, 3, 0), 8, 0));
201:     jPanel1.add(jPanel4, new GridBagConstraints(2, 0, 1, 2, 1.0, 1.0
202:         ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
203:         new Insets(0, 0, 3, 4), -9, 1));
204:     jPanel4.add(jButton_zAxis,
205:         new GridBagConstraints(2, 1, 1, 1, 0.0, 0.0
206:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
207:         new Insets(16, 0, 27, 11), 0, 0));
208:     jPanel4.add(jCheckBox_ShowTumorWindow,
209:         new GridBagConstraints(0, 0, 3, 1, 0.0, 0.0
210:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
211:         new Insets(4, 11, 0, 5), 5, 0));
212:     jPanel4.add(jButton_xAxis,
213:         new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
214:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,

```

```

215:         new Insets(16, 11, 27, 0), 0, 0));
216: jPanel4.add(jRadioButton_yAxis,
217:     new GridBagConstraints(1, 1, 1, 1, 0.0, 0.0
218:     ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
219:     new Insets(16, 0, 27, 0), 0, 0));
220: jPanel1.add(jPanel6, new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0
221:     ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
222:     new Insets(0, 6, 0, 0), 0, 0));
223: jPanel6.add(jProgressBar_cellDoubles,
224:     new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
225:     ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
226:     new Insets(0, 4, 0, 1), 45, 1));
227: jPanel6.add(jProgressBar_popDoubles,
228:     new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
229:     ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
230:     new Insets(0, 4, 1, 1), 45, 1));
231: jPanel1.add(jPanel2, new GridBagConstraints(0, 1, 1, 1, 1.0, 1.0
232:     ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
233:     new Insets(0, 0, 3, 0), 20, 5));
234: jPanel2.add(stopButton, new GridBagConstraints(1, 0, 1, 1, 0.0, 0.0
235:     ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
236:     new Insets(8, 15, 9, 34), 0, 0));
237: jPanel2.add(startButton, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
238:     ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
239:     new Insets(8, 23, 9, 0), 0, 0));
240: jPopupMenu1.addSeparator();
241: jProgressBar_popDoubles.setStringPainted(true);
242: jProgressBar_popDoubles.setMinimum(1);
243: jProgressBar_cellDoubles.setStringPainted(true);
244: jProgressBar_cellDoubles.setMinimum(1);
245: } // *** end jbInit ***
246:
247: // Overridden so we can exit on System Close.
248: protected void processWindowEvent(WindowEvent e) {
249:     super.processWindowEvent(e);
250:     if(e.getID() == WindowEvent.WINDOW_CLOSING) {
251:         System.exit(0);
252:     }
253: }
254:
255: // Start execution of the control panel by placing the simulation's default
256: // values (as received from the calling main application) into the panel's
257: // text fields. Next, the stop button is turned "off" (until the the
258: // simulation is started) and a new window is created. Note that the tumor
259: // tumor window is created but not visible. Visibility is set to false at
260: // the time of window instantiation so that it's not visible until "turned on"
261: // by the user.
262: void run(int xDim, int yDim, int popDoubles, int cellSize) {
263:     jTextField_popDoubles.setText(Integer.toString(popDoubles));
264:     jTextField_cellResolution.setText(Integer.toString(cellSize));
265:     jTextField_windowWidth.setText(Integer.toString(xDim));
266:     jTextField_windowHeight.setText(Integer.toString(yDim));
267:     stopButton.setEnabled(false);
268:     tumorWindow = new TumorWindow(xDim, yDim);
269:     // Configure progress bars.
270:     jProgressBar_popDoubles.setMinimum(1);
271:     jProgressBar_popDoubles.setString("Doubling: ");
272:     jProgressBar_popDoubles.setStringPainted(true);
273:     jProgressBar_cellDoubles.setMinimum(1);
274:     jProgressBar_cellDoubles.setStringPainted(true);
275: }
276:
277: // The following methods performs the specified actions when the control
278: // panel's "start" button is pressed.
279: void startButton_actionPerformed(ActionEvent e) {
280:     // In case the user changed any of the default values for the simulation,
281:     // obtain the values from all text fields in the control panel.
282:     this.popDoubles = Integer.parseInt(this(jTextField_popDoubles.getText());
283:     jProgressBar_popDoubles.setMaximum(popDoubles);
284:     this.cellSize = Integer.parseInt(this(jTextField_cellResolution.getText());
285:     this.windowWidth = Integer.parseInt(jTextField_windowWidth.getText());
286:     this.windowHeight = Integer.parseInt(jTextField_windowHeight.getText());
287:     tumorWindow.setSize(windowWidth, windowHeight);
288:     // Reset window size to a multiple of cell size.

```

```

289:     int windowWidth =
290:         Math.max(tumorWindow.getWidth(), cellSize) / cellSize * cellSize;
291:     int windowHeight =
292:         Math.max(tumorWindow.getHeight(), cellSize) / cellSize * cellSize;
293:     // Calculate the number of cells possible for the x and y directions.
294:     cellWidth = windowWidth / cellSize;
295:     cellHeight = windowHeight / cellSize;
296:     cellDepth = (cellWidth + cellHeight)/2;
297:     // Resize the applet window to a multiple of the scale value.
298:     tumorWindow.setSize(windowWidth, windowHeight);
299:     // Calculate coordinates for the cancer cell (middle of the growth space).
300:     firstCellX = cellWidth / 2;
301:     firstCellY = cellHeight / 2;
302:     firstCellZ = cellDepth / 2;
303:     // Instantiate the cancer tumor in its own thread, pass the parameters,
304:     // and give the thread a name.
305:     cancerTumor = new CancerTumor(popDoubles,
306:                                   cellSize,
307:                                   cellWidth,
308:                                   cellHeight,
309:                                   cellDepth,
310:                                   firstCellX,
311:                                   firstCellY,
312:                                   firstCellZ);
313:     startButton.setEnabled(false); // Simulation started; disallow new start.
314:     stopButton.setEnabled(true); // Simulation started; allow to be stopped.
315:     // Set range of values for tumor color based upon potential size.
316:     tumorWindow.setColorChart(cellWidth);
317:     // *** For output to console only ***
318:     System.out.println("=====");
319:     System.out.println("Tumor growth started with " +
320:                       popDoubles + " population doublings.");
321:     System.out.println("Cell resolution: " + cellSize + " (pixels)");
322:     System.out.println("Window resolution: " + cellWidth +
323:                       " x " + cellHeight + " (cells)");
324:     System.out.println("First cell positioned at " +
325:                       firstCellX + ", " + firstCellY + ", " + firstCellZ);
326:     System.out.println("=====");
327:     // Start the simulation.
328:     cancerTumor.start();
329: }
330:
331: // The following methods performs the specified actions when the control
332: // panel's "stop" button is pressed.
333: void stopButton_actionPerformed(ActionEvent e) {
334:     cancerTumor.stop(); // Suspend execution of the cancer tumor.
335:     startButton.setEnabled(true);
336:     stopButton.setEnabled(false);
337: }
338:
339: // The following method performs the specified actions when the control
340: // panel's "show" button is pressed.
341: void jCheckBox_ShowTumorWindow_actionPerformed(ActionEvent e) {
342:     if (jCheckBox_ShowTumorWindow.isSelected()) {
343:         if (jRadioButton_xAxis.isSelected()) {
344:             jRadioButton_xAxis_actionPerformed(e);
345:         }
346:         else if (jRadioButton_yAxis.isSelected()) {
347:             jRadioButton_yAxis_actionPerformed(e);
348:         }
349:         else if (jRadioButton_zAxis.isSelected()) {
350:             jRadioButton_zAxis_actionPerformed(e);
351:         }
352:         tumorWindow.setVisible(true);
353:         jRadioButton_xAxis.setEnabled(true);
354:         jRadioButton_yAxis.setEnabled(true);
355:         jRadioButton_zAxis.setEnabled(true);
356:     }
357:     else {
358:         jRadioButton_xAxis.setEnabled(false);
359:         jRadioButton_yAxis.setEnabled(false);
360:         jRadioButton_zAxis.setEnabled(false);
361:         tumorWindow.setVisible(false);
362:     }

```

```

363:     }
364:
365:     // The following method performs the specified actions when the control
366:     // panel's "X Axis" radio button is pressed.
367:     void jButton_xAxis_actionPerformed(ActionEvent e) {
368:         jButton_xAxis.setSelected(true);
369:         tumorWindow.setTitle(tumorWindow.WINDOW_TITLE + "X Axis");
370:         jButton_yAxis.setSelected(false);
371:         jButton_zAxis.setSelected(false);
372:         tumorWindow.repaint(cancerTumor.getFaceX(),
373:                             cellSize, cellWidth, cellHeight, cellDepth);
374:     }
375:
376:     // The following method performs the specified actions when the control
377:     // panel's "Y Axis" radio button is pressed.
378:     void jButton_yAxis_actionPerformed(ActionEvent e) {
379:         jButton_xAxis.setSelected(false);
380:         jButton_yAxis.setSelected(true);
381:         tumorWindow.setTitle(tumorWindow.WINDOW_TITLE + "Y Axis");
382:         tumorWindow.panel.repaint();
383:         jButton_zAxis.setSelected(false);
384:         tumorWindow.repaint(cancerTumor.getFaceY(),
385:                             cellSize, cellWidth, cellHeight, cellDepth);
386:     }
387:
388:     // The following method performs the specified actions when the control
389:     // panel's "Z Axis" radio button is pressed.
390:     void jButton_zAxis_actionPerformed(ActionEvent e) {
391:         jButton_xAxis.setSelected(false);
392:         jButton_yAxis.setSelected(false);
393:         jButton_zAxis.setSelected(true);
394:         tumorWindow.setTitle(tumorWindow.WINDOW_TITLE + "Z Axis");
395:         tumorWindow.repaint(cancerTumor.getFaceZ(),
396:                             cellSize, cellWidth, cellHeight, cellDepth);
397:     }
398: }

```

Package Name	Class Name	Source Filename
Tumor04	TumorWindow	TumorWindow.java

```

1:  /*
2:  Title.....: TumorWindow
3:  Version.....: 1.1
4:  Copyright...: Copyright (c) 1999
5:  Author.....: Capt Bruce C Jenkins (GOA-00M)
6:  Company.....: Air Force Institute of Technology
7:  Description: This class is designed to be the main point of control for the
8:  cancer simulation. It deviates from previously developed classes in that it
9:  makes use of Java Swing user interface (UI) componenets. The control panel
10: receives the simulation defaults from the main application and uses them to
11: initiate tumor growth.
12:
13: Modification history:
14:
15: Version 1.0 (10 Dec 1999)
16: - Receive a projection (matrix) from the calling method. Determine the range of
17: values for the display color and paint the screen.
18:
19: Version 1.1 (27 Jan 2000)
20: - The initial version could not display small tumors. The range of values
21:   chosen was based on the range "possible," which was derived from the size of
22:   the cube.
23: - This version is modified to adjust the color scale based upon the actual tumor
24:   size.
25: */
26:
27: package Tumor04;

```

```

28:
29: import java.awt.*;
30: import javax.swing.*;
31:
32: class TumorWindow extends JFrame {
33:
34:     private Color    colorChart[];        // Holds range of color values.
35:     final String    WINDOW_TITLE = "Tumor Projection: ";
36:     JPanel    panel;
37:     private static int cellSize;          // Pixel resolution of cancer cell.
38:     private static int cellWidth;         // Width, in cells, of growth space.
39:     private static int cellHeight;        // Height, in cells, of growth space.
40:     private static int cellDepth;        // Depth, in cells, of growth space.
41:     int[][] face;                         // Projection facing.
42:
43:     // Create the window in which a growing tumor will be shown.
44:     // Inputs: The default window size
45:     // Return: n/a
46:     public TumorWindow(int width, int height) {
47:         panel = new JPanel();             // Create a new panel instance.
48:         this.setSize(width, height);      // Set the default window size.
49:         this.setResizable(false);         // Don't allow resizing.
50:     }
51:
52:     // Create a table of Java Color objects by determining the range of colors to
53:     // use, and then assigning a color value to each possible value in the range.
54:     // Inputs: Tumor density + 1.
55:     // Return: n/a
56:     public void setColorChart(int density) {
57:         int cScale = (256/density);        // Valid color range is 0-255.
58:         colorChart = new Color[density];   // Create array of length = density.
59:         // Fill color chart with color values that are multiples of scale. The net
60:         // effect is the use of the full range of the color scale for max contrast.
61:         // We are using shades of white to represent the tumor density, so there is
62:         // no need to differentiate between the red, blue, and green arguments
63:         // passed to the Color() constructor. Instead, we will use a single,
64:         // identical value for all arguments.
65:         int c = 0; // Initial rgb value. Color(0,0,0) ==> black.
66:         // Construct the color chart. No cells in the projection will show as
67:         // black. Max cells in the projection will show as white. The number of
68:         // cells between 0 and Max will be displayed as shades of gray from black
69:         // to white by steps of cScale.
70:         // For example:
71:         // density = 8 (0-7 cancer cells)
72:         // cScale = 256/8 = 32
73:         // colorChart = {{0,0,0},{32,32,32},{64,64,64},{96,96,96},{128,128,128},
74:         //                {160,160,160},{192,192,192},{224,224,224}}
75:         for (int i = 0; i < density; i++) {
76:             colorChart[i] = new Color(c, c, c);
77:             c += cScale;
78:         }
79:     }
80:
81:     // Paint the projection within the window.
82:     // Inputs: Cell resolution and growth space dimensions.
83:     // Return: n/a
84:     public void repaint(int[][] face, int cellSize,
85:                        int cellWidth,
86:                        int cellHeight,
87:                        int cellDepth) {
88:         this.face = face;
89:         this.cellSize = cellSize;
90:         this.cellWidth = cellWidth;
91:         this.cellHeight = cellHeight;
92:         this.cellDepth = cellDepth;
93:         // Determine new ColorChart values.
94:         int maxDensity = 0;
95:         for (int i = 0; i < cellHeight; i++) {
96:             for (int j = 0; j < cellWidth; j++) {
97:                 if (this.face[i][j] > maxDensity) {
98:                     maxDensity = this.face[i][j];
99:                 }
100:             }
101:         }

```



```

102:         setColorChart(maxDensity + 1); // Add 1 to account for a density of 0.
103:         this.repaint();
104:     }
105:
106:     // Call this method to reduce screen flickering.
107:     public void update(Graphics g) {
108:         paint(g);
109:     }
110:
111:
112:     // Repaint the screen based upon values in the projection matrix ("face").
113:     public void paint(Graphics g) {
114:         int cellValue;
115:         for (int j = 0; j < cellHeight; j++) {
116:             for (int k = 0; k < cellDepth; k++) {
117:                 cellValue = face[j][k]; // Get value of cube face.
118:                 g.setColor(colorChart[cellValue]);
119:                 g.fillRect(j * cellSize, k * cellSize, cellSize, cellSize);
120:             }
121:         }
122:     }
123:
124:     private void jbInit() throws Exception {
125:         this.setVisible(false); // Make visible when the user wants to see it.
126:     }
127: }

```

Appendix B: Java Source Code - Tumor04_2

Package Name	Class Name	Source Filename
Tumor04_2	CancerSimApp	CancerSimApp.java

```

1: /*
2: Title.....: CancerSim
3: Copyright...: Copyright (c) 1999
4: Author.....: Capt Bruce C Jenkins (GOA-00M)
5: Company.....: Air Force Institute of Technology
6: Description: CancerSim simulates the growth of a 3-dimensional breast cancer
7: tumor. This model utilizes a static l x m x n cube for the tumor growth space.
8: The dimensions of the growth space depend on the pixel dimensions chosen for the
9: applet and the scale (size) of the cancer cell in pixels. After initialization,
10: a single cancer cell is positioned approximately in the center of the cube. The
11: cell's position is tracked with a dynamic, object-oriented container (Vector)
12: that is created at runtime. The number of times the container is examined
13: represents the number of population doublings that occur. For each cancer cell
14: found in the vector ("cellVector"), a new cancer cell is placed in an unoccupied
15: position in the cube ("growthSpace"). The direction in which a search for a
16: free space is conducted (there are 26 possible directions) is determined by a
17: uniform random number, which is generated each time a cell must be positioned.
18: (See the written thesis for additional documentation.) The simulation ends when
19: the number of population doublings is complete or the tumor growth has exceeded
20: the bounds of the growthSpace.
21:
22: Modification history:
23:
24: Version 4.0 (16 Dec 1999):
25: 1 In the previous version, cellWidth x cellWidth color objects were created
26: each time the tumor was painted to the screen. There was a significant amount
27: of memory and time overhead associated with this method. In this version of
28: CancerSim, the number of color objects required is reduced by
29: cellWidth*cellWidth. After the number of cells (of the face of the cube) are
30: determined, a color chart is created that contains the range of color shades
31: scaled to the maximum possible number of cells along any of the three axis.
32:
33: 2. The above code was moved to a new class: ControlPanel. This modification is
34: part of a complete restructuring of the program to take advantage of the Java
35: Swing componenets.
36:
37: 3. The applet was converted to an application for the same reason identified
38: above.
39:
40: Version 4.1:
41: The CancerTumor class was modified to remove the use of the 3-D growth space to
42: determine whether or not a candidate cell position was unoccupied. Instead of
43: doing a query on the cube, the cell vector is repeatedly traversed.
44:
45: Version 4.2 (18 Jan 2000):
46: Because of exponential growth in the time required for population doublings in
47: version 4.2, the growth space cube was returned to the CancerTumor class. In
48: this version, all references to "population doublings" were changed to "elapsed
49: time."
50: */
51:
52: package Tumor04_2;
53:
54: import java.awt.*;
55: import javax.swing.UIManager;
56:
57: public class CancerSimApp {
58:
59:     static final String VERSION = "4.2";
60:
61:     //***** Simulation Defaults *****
62:     private final int DEFAULT_X_DIM = 400;           // Width and height of of tumor

```

```

63: private final int DEFAULT_Y_DIM = 400;          // display window.
64: private final int DEFAULT_TIME = 400;           // Elapsed time (units).
65: private final int DEFAULT_SCALE = 6;            // Display cell size.
66: //*****
67:
68: static ControlPanel controlPanel;               // Simulation control component.
69:     boolean    packFrame = false; // Default packing of frames.
70:
71: // Construct the application.
72: // Inputs: None
73: // Return: n/a
74: public CancerSimApp() {
75:     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
76:     controlPanel = new ControlPanel(); // Create a new control panel object.
77:     // Pack frames that have useful preferred size info. For example, from
78:     // their layout.
79:     if (packFrame) {
80:         controlPanel.pack();
81:     }
82:     else {
83:         controlPanel.validate(); // Validate frames that have preset sizes.
84:     }
85:     // Center control panel horizontally and just below the center of the
86:     // vertically screen.
87:     controlPanel.setLocation((screenSize.width - controlPanel.getWidth())/2,
88:                             screenSize.height/2);
89:     controlPanel.setVisible(true);
90:     // Start control panel with defaults.
91:     controlPanel.run(DEFAULT_X_DIM, DEFAULT_Y_DIM,
92:                     DEFAULT_TIME, DEFAULT_SCALE);
93: }
94:
95: // Main method. Establish the application's look and feel and instantiate
96: // the application. For consistency from platform to platform, the look
97: // and feel is set to "Metal."
98: // Inputs: None
99: // Return: n/a
100: public static void main(String[] args) {
101:     try {
102:         UIManager.setLookAndFeel("javax.swing.plaf.motif.MetalLookAndFeel");
103:     }
104:     catch(Exception e) {
105:     }
106:     CancerSimApp cancerSim = new CancerSimApp();
107: }
108: }

```

Package Name	Class Name	Source Filename
Tumor04_2	CancerCell	CancerCell.java

```

1:  /*
2:  Title.....: CancerCell
3:  Version.....: 3.2
4:  Copyright...: Copyright (c) 1999
5:  Author.....: Capt Bruce C Jenkins (GOA-00M)
6:  Company.....: Air Force Institute of Technology
7:  Description: CancerCell is a key components of this simulation. It is
8:  intended that this class have the capability to scale with the increased
9:  complexity inherent in improving the CancerSim application.
10:
11: Modification history:
12: Version 2.0 (10 Nov 1999):
13: The type declaration for "location" has been replace by a 3-dimensional
14: version. This improves the scaleability of the object as the simulation moves
15: to a 3-D environment for growing the cancer cells.
16:
17: Version 3.0 (??):
18:
19: Version 3.1 (07 Dec 1999):
20: The Point class was replaced with Point3f from Java3D. This was done to
21: allow complex vector calculations between two points in 3D space.
22:
23: Version 3.2 (17 Jan 2000):
24: Add the capability to track the age of the cancer cell, the current phase of
25: cell development, and how long it is in that phase.
26:
27: */
28: package Tumor04_2;
29:
30: import javax.vecmath.*;
31: import java.util.Random;
32: import drasys.or.prob.NormalDistribution;
33:
34: public class CancerCell {
35:
36:     private int    cellID = 1;        // Give each new cell a unique ID.
37:     private Point3f location;          // The xyz coordinates of the cancer cell.
38:     private boolean isAlive;           // Indicates if cell is alive or dead.
39:     private int    age;                // Age of cancer cell.
40:     private int    timeInPhase;        // Time to spend in current growth phase.
41:     private int    maxTimeInPhase[] = {10, 0, 0, 0, 0};
42:     private byte   growthPhase;        // The phase of cancer cell growth.
43:                                         // 0 ==> Quiescence (G0).
44:                                         // 1 ==> Gap 1 (G1).
45:                                         // 2 ==> Synthesis (S ).
46:                                         // 3 ==> Gap 2 (G2).
47:                                         // 4 ==> Mytosis (M ).
48:     private NormalDistribution timeDist;
49:
50:
51:     // Create the constuctor so that coordinates are required at instantiation.
52:     // Inputs: Coordinates for new cancer cell
53:     // Return: n/a
54:     CancerCell(float xPos, float yPos, float zPos) {
55:         location = new Point3f(xPos, yPos, zPos);
56:         isAlive = true;
57:         age = 0;
58:         growthPhase = 1;
59:         timeInPhase = 0;
60:         timeDist = new NormalDistribution(12, 0.75);
61:         maxTimeInPhase[1] = (int)Math.round(timeDist.getRandomScaler());
62:     }
63:
64:     // Create constuctor so that no coordinates are required at instantiation.
65:     // Inputs: None
66:     // Return: n/a

```

```

67:  CancerCell() {
68:  }
69:
70:  // Set the current location of the cancer cell.
71:  // Inputs: Coordinates for new cancer cell
72:  // Return: n/a
73:  public void setLocation(float xPos, float yPos, float zPos) {
74:      location.set(xPos, yPos, zPos);
75:  }
76:
77:  // Get the current location of the cancer cell.
78:  // Inputs: None
79:  // Return: Location of cancer cell
80:  public Point3f getLocation() {
81:      return location;
82:  }
83:
84:  // Set the dead/alive condition of the cancer cell. If the cell is alive,
85:  // the ability to divide is assumed.
86:  // Inputs: Boolean value indicating if cell is alive or dead
87:  // Return: n/a
88:  public void setIsAlive(boolean condition) {
89:      isAlive = condition;
90:  }
91:
92:  // Get the dead/alive condition of the cancer cell.
93:  // Inputs: None
94:  // Return: Boolean value indicating if cell is alive or dead
95:  public boolean getIsAlive(){
96:      return isAlive;
97:  }
98:
99:  // Set the age of the cancer cell.
100:  // Inputs: Current age of the cancer cell.
101:  // Return: n/a.
102:  public void setAge(int theAge){
103:      age = theAge;
104:  }
105:
106:  // Get the age of the cancer cell.
107:  // Inputs: None
108:  // Return: Integer value representing the age of the cancer cell.
109:  public int getAge(){
110:      return age;
111:  }
112:
113:  // Increment the time in current phase by one time unit.
114:  // Inputs: None
115:  // Return: n/a.
116:  public void setTimeInPhase(int theTime){
117:      timeInPhase = theTime;
118:  }
119:
120:  // Get the number of time unit spent in the current phase.
121:  // Inputs: None
122:  // Return: Integer value representing the time spent in the current phase.
123:  public int getTimeInPhase(){
124:      return timeInPhase;
125:  }
126:
127:  // Set the growth phase of the cancer cell. If the cancer cell is new, then
128:  // set the phase to Gap 1.
129:  // Inputs: None.
130:  // Return: n/a.
131:  public void setGrowthPhase(byte thePhase){
132:      growthPhase = thePhase;
133:      timeInPhase = 0;
134:  }
135:
136:  // Get the growth phase of the cancer cell.
137:  // Inputs: None.
138:  // Return: Integer value representing the growth phase of the cancer cell.
139:  public int getGrowthPhase(){
140:      return growthPhase;

```

```

141:  }
142:
143:  // Set the maximum time allowed for a given phase.
144:  // Inputs: The phase and maximum time allowed.
145:  // Return: n/a.
146:  public void setMaxTimeInPhase(int phase, int maxTime) {
147:      this.maxTimeInPhase[phase] = maxTime;
148:  }
149:
150:  // Get the maximum time allowed for a given phase.
151:  // Inputs: None.
152:  // Return: Integer value representing the max time allowed in current phase.
153:  public int getMaxTimeInPhase(int phase) {
154:      return this.maxTimeInPhase[phase];
155:  }
156:
157:  // Give the cancer cell a unique ID.
158:  // Inputs: Cell ID.
159:  // Return: n/a.
160:  public void setCellID (int newID) {
161:      this.cellID = newID;
162:  }
163:
164:  // Get the unique cancer cell ID.
165:  // Inputs: None.
166:  // Return: Integer value representing a unique cell ID.
167:  public int getCellID () {
168:      return this.cellID;
169:  }
170:
171:  // Start the process of cell division. While in the state of dividing, a new
172:  // cancer cell is created. A direction of travel is determined by the tumor.
173:  // Inputs: None
174:  // Return: New cancer cell with coordinates of the current cell.
175:  public CancerCell divide() {
176:      // Put self back into phase G1.
177:      this.growthPhase = 1;
178:      timeDist = new NormalDistribution(12, 0.75);
179:      this.maxTimeInPhase[1] = (int)Math.round(timeDist.getRandomScaler());
180:      this.timeInPhase = 0;
181:      // Get current position and pass to new cell.
182:      float x = this.location.x;
183:      float y = this.location.y;
184:      float z = this.location.z;
185:      return new CancerCell(x, y, z);
186:  }
187: }
188:

```

Package Name	Class Name	Source Filename
Tumor04_2	CellDirection	CellDirection.java

{ See Appendix A, CellDirection.java }

Package Name	Class Name	Source Filename
Tumor04_2	CancerTumor	CancerTumor.java

```

1: /*
2: Title.....: CancerTumor
3: Version.....: 4.0
4: Copyright...: Copyright (c) 1999
5: Author.....: Capt Bruce C Jenkins (GOA-00M)

```

```

6: Company....: Air Force Institute of Technology
7: Description: This class is implemented as the container for all cancer cells
8: created in this simulation. It is intended to be the "brains" of the collection
9: of cells. It will keep track of the number of cells that exist, control when
10: and how many times a cell may divide, their rate of mitosis (death), as well as
11: other factors influencing the behavior of the cells as a collective entity.
12:
13: Modification history:
14: Version 1.0 (10 Nov 1999):
15: This initial version of CancerTumor replaces the "cellVector" in CancerSim v2.
16: Previously, the intent was to create a CancerTumor "containing" a cellVector
17: object. After additional research, it makes better sense to simply code this
18: class so that CancerTumor, in essence, "is" the cellVector. Because of as yet
19: unforeseen embellishments, this object will be designed to run in its own thread.
20: If it turns out that a separate thread is not required, this feature is easily
21: removed from the object.
22:
23: Version 2.0 (Dec 1999):
24: The CancerTumor was changed to a "has-a" relationship with the CellVector class.
25: It turns out that this was a more flexible option than what was previously
26: implemented.
27:
28: Version 3.0 (09 Jan 2000)
29: This version removed the growthSpace cube and checked for candidate cell space
30: directly in cellVector. The removal of the cube significantly increased the
31: execution time of the simulation.
32:
33: Version 4.0 (17 Jan 2000):
34: This version reverts back to the 2.0 model, which uses the growthSpace cube to
35: check for free space. In addition, the following enhancements were made:
36: - Implement use of the stopTumor flag through the public method setStop().
37: - A flag check was added at the point of updating the cellDoubles progress bar.
38:   If the flage is set, the program breaks out of the tumor growth loops, prints
39:   final values for tumor growth distances and terminates.
40: - Converted growthSpace to a cube of integers. Since we're summing to the face
41:   of the cube, the maximum value allowed by a byte would mean a maximum of 23
42:   population doublings.
43:
44: */
45: package Tumor04_2;
46:
47: import java.util.*;
48: import java.awt.*;
49: import javax.vecmath.*;
50: import java.awt.Toolkit;
51: import Tumor04_2.CellRules.*;
52:
53: class CancerTumor extends Thread {
54:
55:     private int      cellCount = 0; // Number of cells in the cancer tumor.
56:     private int      cellsAlive = 0; // Cells living cells at end of simulation.
57:     private boolean  stopTumor = false; // Flag to stop tumor growth.
58:     private int      simTime; // Number of times cell pop may divide.
59:     private int      cellSize; // Cell size (resolution) in pixels.
60:     private int      growthSpace [][][]; // Tumor growth space (cube).
61:     private int      cubeWidth; // Width, in cells, of the growth space.
62:     private int      cubeHeight; // Height, in cells, of the growth space.
63:     private int      cubeDepth; // Depth, in cells, of the growth space.
64:     private int      xPos; // X-axis position of cell in growthSpace.
65:     private int      yPos; // Y-axis position of cell in growthSpace.
66:     private int      zPos; // Z-axis position of cell in growthSpace.
67:     private CellDirection cellDirection;
68:     private CellRules cellRules;
69:     private Point3f tPoint; // Temporary point.
70:     private Point3f tumorOrigin; // Track origin of cancer tumor.
71:     private Vector tumor; // Container to manage cancer cells.
72:     private float cellDistance = 0; // The next four attributes are used to
73:     private float sumDistance = 0; // keep track of distance metrics within
74:     private float meanDistance = 0; // the tumor.
75:     private float maxDistance = 0;
76:     private Date currentTime; // The next four attributes are used to
77:     private long startTime; // keep track of time metrics for the
78:     private long stopTime; // growth of cancer cells with the tumor.

```

```

80:     private long     elapsedTime;
81:     private DataManagement timeData = new DataManagement();
82:     private Collection timeList = new ArrayList(); // List of time stamps.
83:
84:
85:     // Construct the tumor by creating a new cancer cell at position x, y, z.
86:     // Inputs: Coordinates of new cancer cell
87:     // Return: n/a
88:     CancerTumor (int simTime,
89:                 int cellSize,
90:                 int cellWidth,
91:                 int cellHeight,
92:                 int cellDepth,
93:                 int FirstCellX,
94:                 int FirstCellY,
95:                 int FirstCellZ) {
96:         // Set initial tumor capacity to 512; doubles when needed.
97:         tumor = new Vector(512);
98:         this.simTime = simTime;
99:         this.cellSize = cellSize;
100:        this.cubeWidth = cellWidth;
101:        this.cubeHeight = cellHeight;
102:        this.cubeDepth = cellDepth;
103:        this.xPos = FirstCellX;
104:        this.yPos = FirstCellY;
105:        this.zPos = FirstCellZ;
106:    }
107:
108:    // Initialize the growth space cube to zeros, place first cell and update
109:    // all three facings.
110:    // Inputs: None
111:    // Return: n/a
112:    private void init() {
113:        this.setStop(false);
114:        // Create the cube.
115:        this.growthSpace = new int [cubeWidth][cubeHeight][cubeDepth];
116:        // Initialize the cube.
117:        for (int i = 0; i < cubeWidth; i++) {
118:            for (int j = 0; j < cubeHeight; j++) {
119:                for (int k = 0; k < cubeDepth; k++) {
120:                    growthSpace[i][j][k] = 0; // Set each cell to "white."
121:                }
122:            }
123:        }
124:        // Place first cancer cell in growth space.
125:        growthSpace[xPos][yPos][zPos] = 1;
126:        // Update faces of the cube.
127:        growthSpace[xPos][yPos][0] = 1;
128:        growthSpace[xPos][0][zPos] = 1;
129:        growthSpace[0][yPos][zPos] = 1;
130:        // Place the first cancer cell in the tumor.
131:        tumor.addElement(new CancerCell(xPos, yPos, zPos));
132:        cellCount++;
133:        cellsAlive++;
134:        // ID the origin of the cancer tumor.
135:        this.tumorOrigin = new Point3f(xPos, yPos, zPos);
136:        // Instantiate the "direction finder."
137:        this.cellDirection = new CellDirection();
138:    }
139:
140:
141:
142:    // Set the cell count to the number of cells currently in the tumor.
143:    // Inputs: None
144:    // Return: n/a
145:    public void setCellCount() {
146:        this.cellCount = tumor.size();
147:    }
148:
149:    // Get the number of cells currently in the tumor.
150:    // Inputs: None
151:    // Return: The number of cells currently in the tumor
152:    public int getCellCount() {
153:        return this.cellCount;

```



```

154:     }
155:
156:     // Set the stop flag.
157:     // Inputs: Condition of flag: set (true) or not (false)
158:     // Return: n/a
159:     public void setStop(boolean condition) {
160:         this.stopTumor = condition;
161:     }
162:
163:     // Get the stop flag.
164:     // Inputs: None
165:     // Return: Condition of flag: set (true) or not (false)
166:     public boolean getStop() {
167:         return this.stopTumor;
168:     }
169:
170:     // Get a copy of the cube representing the X projection.
171:     // Inputs: None
172:     // Return: Projection Matrix
173:     public int[][] getFaceX() {
174:         int [][] face = new int[cubeWidth][cubeDepth];
175:         for (int y = 0; y < this.cubeHeight; y++) {
176:             for (int z = 0; z < this.cubeDepth; z++) {
177:                 face[y][z] = this.growthSpace[0][y][z];
178:             }
179:         }
180:         return face;
181:     }
182:
183:     // Get a copy of the cube representing the Y projection.
184:     // Inputs: None
185:     // Return: Projection Matrix
186:     public int[][] getFaceY() {
187:         int [][] face = new int[cubeHeight][cubeDepth];
188:         for (int x = 0; x < this.cubeWidth; x++) {
189:             for (int z = 0; z < this.cubeDepth; z++) {
190:                 face[x][z] = this.growthSpace[x][0][z];
191:             }
192:         }
193:         return face;
194:     }
195:
196:     // Get a copy of the cube representing the Z projection.
197:     // Inputs: None
198:     // Return: Projection Matrix
199:     public int[][] getFaceZ() {
200:         int [][] face = new int[cubeWidth][cubeHeight];
201:         for (int x = 0; x < this.cubeHeight; x++) {
202:             for (int y = 0; y < this.cubeHeight; y++) {
203:                 face[x][y] = this.growthSpace[x][y][0];
204:             }
205:         }
206:         return face;
207:     }
208:
209:     // This method will be used to execute the tumor on its own thread.
210:     // Inputs: None
211:     // Return: n/a
212:     public void run() {
213:         init();
214:         // For each population doubling, replicate all current cancer cells. New
215:         // daughter cells are set off with a marker so that they are not replicated
216:         // until the next doubling.
217:         int firstCell = 0;
218:         int lastCell = firstCell;
219:         int lastDaughter = lastCell;
220:         int moveX, moveY, moveZ;
221:         CancerCell currentCell, newCell; // Temporary cells.
222:         CancerSimApp.controlPanel.jProgressBar_cellDoubles.setStringPainted(true);
223:         currentTime = new Date();
224:         startTime = currentTime.getTime();
225:         for (int i = 1; i <= simTime; i++) {
226:             CancerSimApp.controlPanel.jProgressBar_simTime.setValue(i);
227:             CancerSimApp.controlPanel.jProgressBar_simTime.setString("Period: "

```

```

228:         + Integer.toString(i));
229:     // Cycle through the cell vector and test each cell for replication.
230:     CancerSimApp.controlPanel.jProgressBar_cellDoubles.setMaximum(lastCell);
231:     for (int j = firstCell; j <= lastCell; j++) {
232:         if (j%100 == 0) {
233:             if (stopTumor) break;
234:             CancerSimApp.controlPanel.jProgressBar_cellDoubles.setValue(j);
235:         }
236:         // Access current cancer cell and determine if its current growth
237:         // phase allows it to divide.
238:         currentCell = (CancerCell)tumor.elementAt(j);
239:         if (cellRules.canDivide(currentCell)) {
240:             // Determine direction of travel for new cell.
241:             tPoint = cellDirection.getDirection();
242:             moveX = (int)tPoint.x;
243:             moveY = (int)tPoint.y;
244:             moveZ = (int)tPoint.z;
245:             // Cause current cell to create a new cell.
246:             newCell = (CancerCell)currentCell.divide();
247:             // New cell's location coincides with current cell. Get the
248:             // location so direction of travel can be used to determine the
249:             // destination.
250:             tPoint = newCell.getLocation();
251:             xPos = (int)tPoint.x;
252:             yPos = (int)tPoint.y;
253:             zPos = (int)tPoint.z;
254:             // Enter the following loop if the growth space is not empty (i.e.,
255:             // it's occupied by a cancer cell) at the current coordinates and t
256:             // offsets (directions of travel). Within the loop, increment the
257:             // offsets by 1 if they are positive, by -1 if negative. If the
258:             // offset is 0, no action is taken.
259:             while (growthSpace[xPos + moveX][yPos + moveY][zPos + moveZ] != 0)
260:                 if (moveX > 0) moveX++; // Continue in positive X direction.
261:                 else if (moveX < 0) moveX--; // Continue in negative X direction
262:                 if (moveY > 0) moveY++; // Continue in positive Y direction.
263:                 else if (moveY < 0) moveY--; // Continue in negative Y direction
264:                 if (moveZ > 0) moveZ++; // Continue in positive Z direction.
265:                 else if (moveZ < 0) moveZ--; // Continue in negative Z direction
266:             } // end while
267:             xPos += moveX; // ==> xPos = xPos + moveX
268:             yPos += moveY; // ==> yPos = yPos + moveY
269:             zPos += moveZ; // ==> zPos = zPos + moveZ
270:             // Place the cancer cell in the growth space.
271:             growthSpace[xPos][yPos][zPos] = 1; // Red cancer cell.
272:             // Update the "facings."
273:             growthSpace[xPos][yPos][0] += 1;
274:             growthSpace[xPos][0][zPos] += 1;
275:             growthSpace[0][yPos][zPos] += 1;
276:             // Update the index to the last daughter cell and add the daughter
277:             // cell to the tumor.
278:             lastDaughter++;
279:             newCell.setLocation(xPos, yPos, zPos);
280:             newCell.setCellID(++cellCount);
281:             tumor.addElement(newCell);
282:             // Use new position of cell to calculate distance from origin.
283:             tPoint.x = xPos;
284:             tPoint.y = yPos;
285:             tPoint.z = zPos;
286:             cellDistance = tPoint.distance(tumorOrigin);
287:             if (cellDistance > maxDistance) maxDistance = cellDistance;
288:             sumDistance += cellDistance;
289:         } // end if
290:         if (!currentCell.getIsAlive()) {
291:             cellsAlive--;
292:             if (cellsAlive == 0) break;
293:         }
294:     } // end for
295:     if ((stopTumor) || (cellsAlive == 0)) break;
296:     // A population doubling is complete. The added daughter cells are now
297:     // part of the set of cancer cells that are candidates for the next
298:     // population doubling. Update the index of cancer cells to include the
299:     // new daughter cells.
300:     lastCell = lastDaughter;
301:     currentTime = new Date();

```

```

302:         stopTime = currentTime.getTime();
303:         // elapsedTime = stopTime - startTime;
304:         timeList.add(new Long(stopTime - startTime));
305:     } // end for
306:     if (!stopTumor) {
307:         CancerSimApp.controlPanel.jProgressBar_cellDoubles.setValue(lastCell);
308:         CancerSimApp.controlPanel.jProgressBar_simTime.setString("Done!");
309:         timeData.printToFile(timeList);
310:     }
311:     else System.out.println("*** Tumor growth interrupted!");
312:     Toolkit.getDefaultToolkit().beep();
313:     CancerSimApp.controlPanel.stopButton.setEnabled(false);
314:     CancerSimApp.controlPanel.startButton.setEnabled(true);
315:     System.out.println("Maximum distance from tumor origin: " +
316:         maxDistance + " units.");
317:     System.out.println("Mean distance from tumor origin...: " +
318:         sumDistance / cellCount + " units.");
319:     System.out.println("Cell count: " + tumor.size());
320: } // end run
321: }

```

Package Name	Class Name	Source Filename
Tumor04_2	ControlPanel	ControlPanel.java

```

1: /*
2: Title.....: ControlPanel
3: Version....: 1.1
4: Copyright...: Copyright (c) 1999, 2000
5: Author.....: Capt Bruce C Jenkins (GOA-00M)
6: Company.....: Air Force Institute of Technology
7: Description: This class is designed to be the main point of control for the
8: cancer simulation. It deviates from previously developed classes in that it
9: makes use of Java Swing user interface (UI) componenets. The control panel
10: receives the simulation defaults from the main application and uses them to
11: initiate tumor growth.
12:
13: Modification history:
14:
15: Version 1.0 (10 Dec 1999):
16: - The initial verion provide basic funtionality. The user could start and stop
17: the tumor growth, adjust the simulation parameters (population doublings, cell
18: resolution, and window size), display the projection and choose the axis of
19: the projection.
20:
21: Version 1.1:
22: - When the stop button is pressed, instead of using the stop() method to halt
23: the thread (a method that is deprecated), CancerTumor.stopTumor() is called.
24: - All references to "population doublings" were changed to (simulation) "time."
25: */
26:
27: package Tumor04_2;
28:
29: import java.awt.*;
30: import java.awt.event.*;
31: import javax.swing.*;
32: import javax.swing.border.*;
33: import javax.swing.event.*;
34: import com.borland.jbcl.layout.*;
35:
36: public class ControlPanel extends JFrame {
37:
38:     private int simTime; // Number of time units to run simulation.
39:     private int cellSize; // Display cell size.
40:     private int cellWidth, cellHeight, cellDepth; // Cube dimensions.
41:     private int windowWidth, windowHeight; // Window dimensions.
42:     private int firstCellX, firstCellY, firstCellZ; // Tumor origin.
43:

```

```

44:     private CancerTumor cancerTumor;
45:     private TumorWindow tumorWindow;
46:
47:     // The objects below are the primary components of the control panel--the
48:     // applications's graphical user interface.
49:     JPanel jPanel1 = new JPanel();
50:     JButton startButton = new JButton();
51:     JButton stopButton = new JButton();
52:     JPanel jPanel2 = new JPanel();
53:     JPanel jPanel3 = new JPanel();
54:     JTextField jTextField_simTime = new JTextField();
55:     JLabel jLabel1 = new JLabel();
56:     TitledBorder titledBorder1;
57:     TitledBorder titledBorder2;
58:     TitledBorder titledBorder3;
59:     JLabel jLabel2 = new JLabel();
60:     JTextField jTextField_cellResolution = new JTextField();
61:     JCheckBox jCheckBox_ShowTumorWindow = new JCheckBox();
62:     BorderLayout borderLayout1 = new BorderLayout();
63:     JPanel jPanel4 = new JPanel();
64:     JRadioButton jRadioButton_xAxis = new JRadioButton();
65:     JRadioButton jRadioButton_yAxis = new JRadioButton();
66:     JRadioButton jRadioButton_zAxis = new JRadioButton();
67:     JPanel jPanel6 = new JPanel();
68:     JProgressBar jProgressBar_simTime = new JProgressBar();
69:     JProgressBar jProgressBar_cellDoubles = new JProgressBar();
70:     JPopupMenu jPopupMenu1 = new JPopupMenu();
71:     JLabel jLabel3 = new JLabel();
72:     JTextField jTextField_windowWidth = new JTextField();
73:     JTextField jTextField_windowHeight = new JTextField();
74:     JLabel jLabel4 = new JLabel();
75:     JLabel jLabel5 = new JLabel();
76:     GridBagLayout gridBagLayout3 = new GridBagLayout();
77:     GridBagLayout gridBagLayout1 = new GridBagLayout();
78:     GridBagLayout gridBagLayout4 = new GridBagLayout();
79:     GridBagLayout gridBagLayout5 = new GridBagLayout();
80:     GridBagLayout gridBagLayout2 = new GridBagLayout();
81:
82:     // Construct the frame
83:     public ControlPanel() {
84:         enableEvents(AWTEvent.WINDOW_EVENT_MASK);
85:         try {
86:             jbInit();
87:         }
88:         catch(Exception e) {
89:             e.printStackTrace();
90:         }
91:     }
92:
93:     // Component initialization. The method jbInit is required by visual
94:     // designer subsystem of Borland's JBuilder (by Inprise Corporation).
95:     private void jbInit() throws Exception {
96:     /* This section would not work properly when deployed to a jar file. (It
97:        works prior to deployment.) The intent was to replace the Java icon
98:        located in the upper left corner of the control panel with the AFIT logo.
99:        // Add the AFIT shield to the application pane. Note that the file in the
100:        // "getResource(<file>)" method must be in the class directory and also
101:        // must be deployed to the jar or zip file.
102:        ImageIcon icon = new ImageIcon(getClass().getResource("Tumor04_2/AFIT.jpg"));
103:        this.setIconImage(icon.getImage());
104:    */
105:        titledBorder1 = new TitledBorder("Parameters");
106:        titledBorder2 = new TitledBorder("Projection");
107:        titledBorder3 = new TitledBorder("Progress");
108:        setSize(new Dimension(529, 154));
109:        setTitle("CancerSim");
110:        setResizable(false);
111:        this.getContentPane().setLayout(borderLayout1);
112:        jPanel1.setLayout(gridBagLayout5);
113:        startButton.setText("Start");
114:        startButton.addActionListener(new java.awt.event.ActionListener() {
115:            public void actionPerformed(ActionEvent e) {
116:                startButton_actionPerformed(e);
117:            }

```

```

118:     });
119:     stopButton.setText("Stop");
120:     stopButton.addActionListener(new java.awt.event.ActionListener() {
121:         public void actionPerformed(ActionEvent e) {
122:             stopButton_actionPerformed(e);
123:         }
124:     });
125:     jPanel2.setLayout(gridBagLayout3);
126:     jPanel3.setLayout(gridBagLayout1);
127:     jTextField_simTime.setText("");
128:     jTextField_simTime.setHorizontalAlignment(SwingConstants.RIGHT);
129:     jLabel1.setFont(new java.awt.Font("SansSerif", 0, 12));
130:     jLabel1.setHorizontalAlignment(SwingConstants.RIGHT);
131:     jLabel1.setText("Time Periods:");
132:     jPanel3.setBorder(titledBorder1);
133:     jLabel2.setText("Cell Resolution:");
134:     jLabel2.setFont(new java.awt.Font("SansSerif", 0, 12));
135:     jLabel2.setHorizontalAlignment(SwingConstants.RIGHT);
136:     jTextField_cellResolution.setText("");
137:     jTextField_cellResolution.setHorizontalAlignment(SwingConstants.RIGHT);
138:     jCheckBox_ShowTumorWindow.setText("Show Tumor");
139:     jCheckBox_ShowTumorWindow.addActionListener(new java.awt.event.ActionListener() {
140:         public void actionPerformed(ActionEvent e) {
141:             jCheckBox_ShowTumorWindow_actionPerformed(e);
142:         }
143:     });
144:     jPanel4.setLayout(gridBagLayout4);
145:     jPanel4.setBorder(titledBorder2);
146:     jRadioButton_xAxis.setText("X");
147:     jRadioButton_xAxis.setSelected(true);
148:     jRadioButton_xAxis.setEnabled(false);
149:     jRadioButton_yAxis.setEnabled(false);
150:     jRadioButton_zAxis.setEnabled(false);
151:     jCheckBox_ShowTumorWindow.setEnabled(false);
152:     jRadioButton_xAxis.addActionListener(new java.awt.event.ActionListener() {
153:         public void actionPerformed(ActionEvent e) {
154:             jRadioButton_xAxis_actionPerformed(e);
155:         }
156:     });
157:     jRadioButton_yAxis.setText("Y");
158:     jRadioButton_yAxis.addActionListener(new java.awt.event.ActionListener() {
159:         public void actionPerformed(ActionEvent e) {
160:             jRadioButton_yAxis_actionPerformed(e);
161:         }
162:     });
163:     jRadioButton_zAxis.setText("Z");
164:     jRadioButton_zAxis.addActionListener(new java.awt.event.ActionListener() {
165:         public void actionPerformed(ActionEvent e) {
166:             jRadioButton_zAxis_actionPerformed(e);
167:         }
168:     });
169:     jPanel6.setLayout(gridBagLayout2);
170:     jPanel6.setBorder(titledBorder3);
171:     jLabel3.setHorizontalAlignment(SwingConstants.RIGHT);
172:     jLabel3.setFont(new java.awt.Font("SansSerif", 0, 12));
173:     jLabel3.setText("Window Resolution:");
174:     jTextField_windowWidth.setText("");
175:     jTextField_windowWidth.setHorizontalAlignment(SwingConstants.RIGHT);
176:     jTextField_windowHeight.setText("");
177:     jTextField_windowHeight.setHorizontalAlignment(SwingConstants.RIGHT);
178:     jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
179:     jLabel4.setText("X");
180:     jLabel5.setText("Y");
181:     jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
182:     this.getContentPane().add(jPanel1, BorderLayout.CENTER);
183:     jPanel1.add(jPanel3, new GridBagConstraints(1, 0, 1, 2, 1.0, 1.0,
184:         GridBagConstraints.CENTER, GridBagConstraints.BOTH,
185:         new Insets(0, 0, 3, 0), -5, -5));
186:     jPanel3.add(jLabel1, new GridBagConstraints(0, 0, 2, 1, 0.0, 0.0,
187:         GridBagConstraints.WEST, GridBagConstraints.NONE,
188:         new Insets(0, 47, 0, 0), 6, 0));
189:     jPanel3.add(jLabel3, new GridBagConstraints(0, 2, 2, 1, 0.0, 0.0,
190:         GridBagConstraints.WEST, GridBagConstraints.NONE,

```

```

191:         new Insets(10, 6, 0, 0), 14, 0));
192:     jPanel3.add(jTextField_windowWidth,
193:         new GridBagConstraints(1, 3, 1, 1, 1.0, 0.0
194:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
195:         new Insets(0, 0, 4, 15), 25, 0));
196:     jPanel3.add(jLabel2, new GridBagConstraints(0, 1, 2, 1, 0.0, 0.0
197:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
198:         new Insets(12, 16, 0, 0), 26, 0));
199:     jPanel3.add(jLabel4, new GridBagConstraints(0, 3, 1, 1, 0.0, 0.0
200:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
201:         new Insets(0, 72, 4, 0), 7, 0));
202:     jPanel3.add(jLabel5, new GridBagConstraints(2, 3, 1, 1, 0.0, 0.0
203:         ,GridBagConstraints.WEST, GridBagConstraints.NONE,
204:         new Insets(0, 0, 4, 0), 8, 0));
205:     jPanel3.add(jTextField_windowHeight,
206:         new GridBagConstraints(3, 3, 1, 1, 1.0, 0.0
207:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
208:         new Insets(0, 0, 4, 2), 25, 0));
209:     jPanel3.add(jTextField_cellResolution,
210:         new GridBagConstraints(3, 1, 1, 1, 1.0, 0.0
211:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
212:         new Insets(9, 0, 0, 2), 25, 0));
213:     jPanel3.add(jTextField_simTime,
214:         new GridBagConstraints(3, 0, 1, 1, 1.0, 0.0
215:         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
216:         new Insets(0, 0, 0, 2), 25, 0));
217:     jPanel1.add(jPanel4, new GridBagConstraints(2, 0, 1, 2, 1.0, 1.0
218:         ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
219:         new Insets(0, 0, 3, 5), 1, 0));
220:     jPanel4.add(jRadioButton_zAxis,
221:         new GridBagConstraints(2, 1, 1, 1, 0.0, 0.0
222:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
223:         new Insets(16, 0, 28, 2), 0, 0));
224:     jPanel4.add(jCheckBox_ShowTumorWindow,
225:         new GridBagConstraints(0, 0, 3, 1, 0.0, 0.0
226:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
227:         new Insets(4, 8, 0, 2), 5, 0));
228:     jPanel4.add(jRadioButton_xAxis,
229:         new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
230:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
231:         new Insets(16, 8, 28, 0), 0, 0));
232:     jPanel4.add(jRadioButton_yAxis,
233:         new GridBagConstraints(1, 1, 1, 1, 0.0, 0.0
234:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
235:         new Insets(16, 0, 28, 0), 0, 0));
236:     jPanel1.add(jPanel6, new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0
237:         ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
238:         new Insets(0, 6, 0, 0), 0, 0));
239:     jPanel6.add(jProgressBar_cellDoubles,
240:         new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
241:         ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
242:         new Insets(0, 4, 1, 3), 37, -2));
243:     jPanel6.add(jProgressBar_simTime,
244:         new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
245:         ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
246:         new Insets(0, 4, 0, 3), 37, -2));
247:     jPanel1.add(jPanel2, new GridBagConstraints(0, 1, 1, 1, 1.0, 1.0
248:         ,GridBagConstraints.CENTER, GridBagConstraints.BOTH,
249:         new Insets(0, 0, 3, 0), 22, 4));
250:     jPanel2.add(stopButton, new GridBagConstraints(1, 0, 1, 1, 0.0, 0.0
251:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
252:         new Insets(8, 15, 9, 34), 0, 0));
253:     jPanel2.add(startButton, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
254:         ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
255:         new Insets(8, 23, 9, 0), 0, 0));
256:     jPopupMenu1.addSeparator();
257:     jProgressBar_simTime.setStringPainted(true);
258:     jProgressBar_simTime.setMinimum(1);
259:     jProgressBar_cellDoubles.setStringPainted(true);
260:     jProgressBar_cellDoubles.setMinimum(1);
261: } // *** end jbInit ***
262:
263: // Overridden so we can exit on System Close.
264: protected void processWindowEvent(WindowEvent e) {

```

```

265:         super.processWindowEvent(e);
266:         if(e.getID() == WindowEvent.WINDOW_CLOSING) {
267:             System.exit(0);
268:         }
269:     }
270:
271:     // Start execution of the control panel by placing the simulation's default
272:     // values (as received from the calling main application) into the panel's
273:     // text fields. Next, the stop button is turned "off" (until the the
274:     // simulation is started) and a new window is created. Note that the tumor
275:     // window is created but not visible. Visibility is set to false at
276:     // the time of window instantiation so that it's not visible until "turned on"
277:     // by the user.
278:     void run(int xDim, int yDim, int simTime, int cellSize) {
279:         jTextField_simTime.setText(Integer.toString(simTime));
280:         jTextField_cellResolution.setText(Integer.toString(cellSize));
281:         jTextField_windowWidth.setText(Integer.toString(xDim));
282:         jTextField_windowHeight.setText(Integer.toString(yDim));
283:         stopButton.setEnabled(false);
284:         tumorWindow = new TumorWindow(xDim, yDim);
285:         // Configure progress bars.
286:         progressBar_simTime.setMinimum(1);
287:         progressBar_simTime.setString("Period: ");
288:         progressBar_simTime.setStringPainted(true);
289:         progressBar_cellDoubles.setMinimum(1);
290:         progressBar_cellDoubles.setStringPainted(true);
291:     }
292:
293:     // The following methods performs the specified actions when the control
294:     // panel's "start" button is pressed.
295:     void startButton_actionPerformed(ActionEvent e) {
296:         // In case the user changed any of the default values for the simulation,
297:         // obtain the values from all text fields in the control panel.
298:         this.simTime = Integer.parseInt(this(jTextField_simTime.getText()));
299:         progressBar_simTime.setMaximum(simTime);
300:         this.cellSize = Integer.parseInt(this(jTextField_cellResolution.getText()));
301:         this.windowWidth = Integer.parseInt(jTextField_windowWidth.getText());
302:         this.windowHeight = Integer.parseInt(jTextField_windowHeight.getText());
303:         tumorWindow.setSize(windowWidth, windowHeight);
304:         // Reset window size to a multiple of cell size.
305:         int windowWidth =
306:             Math.max(tumorWindow.getWidth(), cellSize) / cellSize * cellSize;
307:         int windowHeight =
308:             Math.max(tumorWindow.getHeight(), cellSize) / cellSize * cellSize;
309:         // Calculate the number of cells possible for the x and y directions.
310:         cellWidth = windowWidth / cellSize;
311:         cellHeight = windowHeight / cellSize;
312:         cellDepth = (cellWidth + cellHeight) / 2;
313:         // Resize the applet window to a multiple of the scale value.
314:         tumorWindow.setSize(windowWidth, windowHeight);
315:         // Calculate coordinates for the cancer cell (middle of the growth space).
316:         firstCellX = cellWidth / 2;
317:         firstCellY = cellHeight / 2;
318:         firstCellZ = cellDepth / 2;
319:         // Instantiate the cancer tumor in its own thread, pass the parameters,
320:         // and give the thread a name.
321:         cancerTumor = new CancerTumor(simTime,
322:                                     cellSize,
323:                                     cellWidth,
324:                                     cellHeight,
325:                                     cellDepth,
326:                                     firstCellX,
327:                                     firstCellY,
328:                                     firstCellZ);
329:         startButton.setEnabled(false); // Simulation started; disallow new start.
330:         stopButton.setEnabled(true); // Simulation started; allow to be stopped.
331:         // *** For output to console only ***
332:         System.out.println("=====");
333:         System.out.println("Tumor growth started with " +
334:                             simTime + " time units.");
335:         System.out.println("Cell resolution: " + cellSize + " (pixels)");
336:         System.out.println("Window resolution: " + cellWidth +
337:                             " x " + cellHeight + " (cells)");
338:         System.out.println("First cell positioned at " +

```

```

339:         firstCellX + ", " + firstCellY + ", " + firstCellZ);
340:     System.out.println("=====");
341:     // Start the simulation.
342:     cancerTumor.start();
343:     this.jCheckBox_ShowTumorWindow.setEnabled(true);
344: }
345:
346: // The following methods performs the specified actions when the control
347: // panel's "stop" button is pressed.
348: void stopButton_actionPerformed(ActionEvent e) {
349:     cancerTumor.setStop(true); // Suspend execution of the cancer tumor.
350:     startButton.setEnabled(true);
351:     stopButton.setEnabled(false);
352: }
353:
354: // The following method performs the specified actions when the control
355: // panel's Show tumor checkbox is selected.
356: void jCheckBox_ShowTumorWindow_actionPerformed(ActionEvent e) {
357:     if (jCheckBox_ShowTumorWindow.isSelected()) {
358:         if (jRadioButton_xAxis.isSelected()) {
359:             jRadioButton_xAxis_actionPerformed(e);
360:         }
361:         else if (jRadioButton_yAxis.isSelected()) {
362:             jRadioButton_yAxis_actionPerformed(e);
363:         }
364:         else if (jRadioButton_zAxis.isSelected()) {
365:             jRadioButton_zAxis_actionPerformed(e);
366:         }
367:         tumorWindow.setVisible(true);
368:         jRadioButton_xAxis.setEnabled(true);
369:         jRadioButton_yAxis.setEnabled(true);
370:         jRadioButton_zAxis.setEnabled(true);
371:     }
372:     else {
373:         jRadioButton_xAxis.setEnabled(false);
374:         jRadioButton_yAxis.setEnabled(false);
375:         jRadioButton_zAxis.setEnabled(false);
376:         tumorWindow.setVisible(false);
377:     }
378: }
379: }
380:
381: // The following method performs the specified actions when the control
382: // panel's "X Axis" radio button is pressed.
383: void jRadioButton_xAxis_actionPerformed(ActionEvent e) {
384:     jRadioButton_xAxis.setSelected(true);
385:     tumorWindow.setTitle(tumorWindow.WINDOW_TITLE + "X Axis");
386:     jRadioButton_yAxis.setSelected(false);
387:     jRadioButton_zAxis.setSelected(false);
388:     tumorWindow.repaint(cancerTumor.getFaceX(),
389:         cellSize, cellWidth, cellHeight, cellDepth);
390: }
391:
392: // The following method performs the specified actions when the control
393: // panel's "Y Axis" radio button is pressed.
394: void jRadioButton_yAxis_actionPerformed(ActionEvent e) {
395:     jRadioButton_xAxis.setSelected(false);
396:     jRadioButton_yAxis.setSelected(true);
397:     tumorWindow.setTitle(tumorWindow.WINDOW_TITLE + "Y Axis");
398:     tumorWindow.panel.repaint();
399:     jRadioButton_zAxis.setSelected(false);
400:     tumorWindow.repaint(cancerTumor.getFaceY(),
401:         cellSize, cellWidth, cellHeight, cellDepth);
402: }
403:
404: // The following method performs the specified actions when the control
405: // panel's "Z Axis" radio button is pressed.
406: void jRadioButton_zAxis_actionPerformed(ActionEvent e) {
407:     jRadioButton_xAxis.setSelected(false);
408:     jRadioButton_yAxis.setSelected(false);
409:     jRadioButton_zAxis.setSelected(true);
410:     tumorWindow.setTitle(tumorWindow.WINDOW_TITLE + "Z Axis");
411:     tumorWindow.repaint(cancerTumor.getFaceZ(),
412:         cellSize, cellWidth, cellHeight, cellDepth);

```



```

413:    }
414: }

```

Package Name	Class Name	Source Filename
Tumor04_2	TumorWindow	TumorWindow.java

{ See Appendix A, TumorWindow.java }

Package Name	Class Name	Source Filename
Tumor04_2	CellRules	CellRules.java

```

1: /*
2: Title.....: CellRules
3: Version.....: 1.0
4: Copyright...: Copyright (c) 1999, 2000
5: Author.....: Capt Bruce C Jenkins (GOA-00M)
6: Company.....: Air Force Institute of Technology
7: Description: This class is intended to contain most of the rules that will
8: describe the behavior of individual cancer cells or even the tumor itself.
9:
10: Modification history:
11:
12: Version 1.0 (18 Jan 2000)
13: - This initial version implements a subset of the rules outline by Palmari, et
14:   al, in a paper entitled "Topographical analysis of spatial patterns generated
15:   by a cellular automaton model of the proliferation of a cancer cell line in
16:   vitro."
17: */
18:
19: package Tumor04_2;
20:
21: import drasys.or.prob.NormalDistribution;
22: import java.util.Random;
23:
24: public class CellRules {
25:
26:     // Determine if the cancer cell under consideration is ready to divide.
27:     // Inputs: Cancer cell object.
28:     // Return: Boolean value that "yes," the cell is ready to divid, or "no" it
29:     // is not ready to divide.
30:     static public boolean canDivide(CancerCell theCell) {
31:
32:         int    growthPhase;           // Growth phase of cell under consideration.
33:         boolean cellStatus = false;   // Attribute to be returned by this method.
34:         double  rNum;                 // Random number returned from distribution.
35:
36:         // First check if the cellError! Bookmark not defined. is alive.  If not,
37:         // exit immediately.
38:         if (!theCell.getIsAlive()) {
39:             return cellStatus;
40:         }
41:         else {
42:             // The cell is being "polled" by the system, so increment the age and
43:             // the time spent in the current phase.
44:             int theAge = theCell.getAge();
45:             theCell.setAge(++theAge);
46:             int theTime = theCell.getTimeInPhase();
47:             theCell.setTimeInPhase(++theTime);
48:             growthPhase = theCell.getGrowthPhase();
49:             NormalDistribution myDist;
50:             // Based upon current growth phase, take the appropriate action.
51:             switch (growthPhase) {

```

```

51:         case 1: // Gap 1      (G1).
52:             if (theCell.getTimeInPhase() >= theCell.getMaxTimeInPhase(1)) {
53:                 theCell.setGrowthPhase((byte)2); // Go to S
54:             }
55:             break;
56:         case 2: // Synthesis (S ).
57:             if (theCell.getMaxTimeInPhase(2) == 0) {
58:                 myDist = new NormalDistribution(7, 0.75);
59:                 rNum = Math.round(myDist.getRandomScaler());
60:                 theCell.setMaxTimeInPhase(2, (int)rNum);
61:             }
62:             else if (theCell.getTimeInPhase() >= theCell.getMaxTimeInPhase(2))
63:                 theCell.setGrowthPhase((byte)3); // Go to G2
64:             }
65:             break;
66:         case 3: // Gap 2      (G2).
67:             if (theCell.getMaxTimeInPhase(3) == 0) {
68:                 myDist = new NormalDistribution(8, 0.75);
69:                 rNum = Math.round(myDist.getRandomScaler());
70:                 theCell.setMaxTimeInPhase(3, (int)rNum);
71:             }
72:             else if (theCell.getTimeInPhase() >= theCell.getMaxTimeInPhase(3))
73:                 theCell.setGrowthPhase((byte)4); // Go to M
74:             }
75:             break;
76:         case 4: // Mytosis    (M )
77:             if (theCell.getMaxTimeInPhase(4) == 0) {
78:                 myDist = new NormalDistribution(1.5, 0.5);
79:                 rNum = Math.round(myDist.getRandomScaler());
80:                 theCell.setMaxTimeInPhase(4, (int)rNum);
81:             }
82:             else if (theCell.getTimeInPhase() >= theCell.getMaxTimeInPhase(4))
83:                 rNum = Math.random();
84:                 cellStatus = true;
85:             }
86:             break;
87:     } // end switch
88: } // end else
89:     return cellStatus;
90: } // end canDivide
91: } // end CellRules

```

Package Name	Class Name	Source Filename
Tumor04_2	DataManagement	DataManagement.java

```

1:  /*
2:  Title.....: DataManagement
3:  Version....: 1.0
4:  Copyright...: Copyright (c) 2000
5:  Author.....: Capt Bruce C Jenkins (GOA-00M)
6:  Company.....: Air Force Institute of Technology
7:  Description: This class is designed to handle the I/O for data generated by the
8:  CancerSim application. It is intended that calls to this package be made from
9:  the CancerTumor class, but there is nothing to prevent calls from being made
10: from any other class.
11:
12: Modification history:
13:
14: Version 1.0 (29 Jan 2000)
15: - Class creation and implementation of print method.
16:
17: */
18:
19: package Tumor04_2;
20:
21: import java.io.*;
22: import java.util.*;

```

```

23:
24: public class DataManagement {
25:
26:     private final String FILENAME = "SimData.txt"; // Default data filename.
27:
28:     private File          cancerSimDataFile; // File handle.
29:     private FileOutputStream fileOutputStream; // Output stream handle.
30:     private PrintStream    fileOut; // Print stream handle.
31:     private boolean        fileOK; // Error detection flag.
32:
33:     // Construct the class.
34:     public DataManagement() {}
35:
36:     // Establish the I/O data streams.
37:     // Inputs: None.
38:     // Return: n/a
39:     private void init() {
40:         try { // Open the file.
41:             cancerSimDataFile = new File(FILENAME);
42:             fileOutputStream = new FileOutputStream(cancerSimDataFile);
43:             fileOut = new PrintStream(fileOutputStream);
44:             fileOK = true;
45:         }
46:         catch (IOException e) {
47:             System.out.println("I/O error: " + e.toString());
48:             fileOK = false;
49:         }
50:     }
51:
52:     // Send data to standard output device.
53:     // Inputs: Collection of objects.
54:     // Return: n/a
55:     public void printToFile(Collection dataList) {
56:         init();
57:         Iterator list = dataList.iterator();
58:         if (fileOK) {
59:             while (list.hasNext()) {
60:                 fileOut.print(list.next() + "\t");
61:             }
62:         }
63:         try {
64:             fileOut.close();
65:         }
66:         catch (Exception e) {
67:             System.out.println("Possible file error: " + e.toString());
68:         }
69:     }
70: }

```

Glossary

Average risk: A measure of the chances of getting breast cancer without the presence of any specific factors known to be associated with the disease.

Benign: Not malignant, not cancerous; cannot invade neighboring tissues or spread to other parts of the body.

Benign breast changes: Noncancerous changes in the breast. Benign breast conditions can cause pain, lumpiness, nipple discharge, and other problems.

Biopsy: Removal of a sample of tissue or cells from the body to assist in diagnosis of a disease.

BRCA1 and BRCA2 genes: The principal genes that, when altered, indicate an inherited susceptibility to breast cancer. These gene alterations are present in 80 to 90 percent of hereditary cases of breast cancer.

Breast density: Glandular tissue in the breast common in younger women, making it difficult for mammography to detect breast cancer.

Calcification: The deposition of calcium salts in body tissues. In the breast, it can be associated with either normal or cancerous tissue.

Cancer: A general name for more than 100 diseases in which abnormal cells grow out of control. Cancer cells can invade and destroy healthy tissues, and they can spread through the bloodstream and the lymphatic system to other parts of the body.

Carcinoma: A malignant tumor arising from epithelial cells, which are cells lining the external or internal surfaces of the body. Carcinomas spread to nearby tissues. They may also spread to distant sites such as lung, liver, lymph nodes and bone. See also metastasis

Carcinoma in situ: A malignant tumor that has not yet become invasive but is confined to the layer of cells from which it arose. A form of pre-invasive cancer.

Carcinoma NOS: Invasive ductal carcinoma not otherwise specified. Comprises 70 percent of all breast cancers.

Centigray: A measure of radiation. 1 centigray = 1 rad.

Chemotherapy: The use of medications (drugs) that are toxic to cancer cells. These drugs kill the cells, or prevent or slow their growth.

Chromosome: A body in the cell nucleus carrying genes.

Clinical breast exam: A physical examination by a doctor or nurse of the breast, underarm, and collarbone area, first on one side, then on the other.

Clinical trial: Research conducted with the patient's permission that usually involves a comparison of two or more treatments or diagnostic methods. The aim is to gain better understanding of the underlying disease process and/or methods to treat it.

Computed tomography (CT) scanning: An imaging technique that uses a computer to organize the information from multiple x-ray views and construct a cross-sectional image of areas inside the body.

Computer-aided diagnosis (CAD): The use of special computer programs to scan mammographic images and flag areas that look suspicious.

Core biopsy: The sampling of breast tissue with a needle to give a tiny cylinder or core of tissue for examination by a pathologist.

Cyclic breast changes: Normal tissue changes that occur in response to the changing levels of female hormones during the menstrual cycle. Cyclic breast changes can produce swelling, tenderness, and pain.

Diagnostic mammogram: The use of a breast x-ray to evaluate the breasts of a woman who has symptoms of disease such as a lump, or whose screening mammogram shows an abnormality.

Differentiation: The degree to which a tumour resembles normal tissue. In general, the closer the resemblance, the better the prognosis. Well-differentiated tumours closely resemble normal tissue.

Digital mammography: A technique for recording x-ray images in computer code, which allows the information to enhance subtle, but potentially significant, changes.

Disease-free survival: The time from the primary treatment of the breast cancer to the first evidence of cancer recurrence.

Ducts: Channels that carry body fluids. Breast ducts transport milk from the breast's lobules out to the nipple during breastfeeding.

Ductal carcinoma in situ (DCIS): A form of breast cancer that requires special consideration. It spreads along the ducts of the breast, rather than forming a lump.

Excisional biopsy: The surgical removal (excision) of an abnormal area of tissue, usually along with a margin of healthy tissue, for microscopic examination. Excisional biopsies remove the entire lump from the breast.

False negative (mammograms): Breast x-rays that miss cancer when it is present.

False positive (mammograms): Breast x-rays that indicate the presence of breast cancer when the disease is truly absent.

Familial breast cancer Breast cancer affecting two or more close relatives, especially in premenopausal women. It implies an inherited disposition.

Fat necrosis: Lumps of fatty material that form in response to a bruise or blow to the breast.

Fibroadenoma: Benign breast tumor made up of both structural (fibro) and glandular (adenoma) tissues.

Generalized breast lumpiness: Breast irregularities and lumpiness, commonplace and noncancerous. Sometimes called "fibrocystic disease" or "benign breast disease."

Higher risk (for breast cancer): A measure of the chances of getting breast cancer when factor(s) known to be associated with the disease are present.

Histology: An examination of the structure of a cell by a pathologist.

Hormones: Chemicals produced by various glands in the body, which produce specific effects on specific target organs and tissues.

Hyperplasia: Increased numbers of epithelial cells. If excessive, there is a slightly increased risk of developing subsequent breast carcinoma. Several types of benign breast conditions involve hyperplasia.

Infiltrating cancer: Cancer that has spread to nearby tissue, lymph nodes under the arm, or other parts of the body. (Same as Invasive cancer.)

Intraductal papilloma: A small wartlike growth that projects into a breast duct.

Invasive cancer: Cancer that has spread to nearby tissue, lymph nodes under the arm, or other parts of the body. (Same as Infiltrating cancer.)

Laser beam scanning: a technology being studied in research for breast cancer detection that shines a laser beam through the breast and records the image produced, using a special camera.

LCIS Lobular carcinoma in situ: It is a misnomer that describes a benign process in the breast. It is not a carcinoma. It is usually detected by chance in the course of a breast biopsy for another lesion.

Lobes, lobules, bulbs: Milk-producing tissues of the breast. Each breast's 15 to 20 lobes branches into smaller lobules, and each lobule ends in scores of tiny bulbs. Milk originates in the bulbs and is carried by ducts to the nipple.

Localization biopsy: The use of mammography to locate tissue containing an abnormality that can be detected only on mammograms, so it can be removed for microscopic examination.

Lumpectomy: Surgical removal of a lump from the breast; usually followed by radiation therapy.

Lymphatic system: A system of vessels which drains fluid out of the head, neck and limbs and returns it to the general circulation. The tissues and organs that produce, store, and transport cells that fight infection and disease.

Lymph node: A small collection of tissue along the lymphatic system that acts as a filter. White cells and cancer cells, in particular, collect in lymph nodes. They are found in the neck, the armpit, the groin and many other places. Lymph nodes are also known as glands.

Macrocalcifications: Coarse calcium deposits. They are most likely due to aging, old injuries or inflammations and usually are associated with benign conditions.

Magnetic resonance imaging (MRI): A technique that uses a powerful magnet linked to a computer to create detailed pictures of areas inside the body.

Malignant: A tumour having the capacity to destroy tissue locally, spread, and cause death.

Malignancy: State of being cancerous. Malignant tumors can invade surrounding tissues and spread to other parts of the body.

Mammogram: A soft tissue x-ray of the breast, which may be used to evaluate a lump, or which may be used as a screening test in women with no signs or symptoms of breast cancer.

Mammography: The examination of breast tissue using x-rays. The process of taking a mammogram.

Mastitis: Infection of the breast. Mastitis is most often seen in nursing mothers.

Medical oncologist: A doctor who specialises in the use of chemotherapy and hormone therapy.

Metastasis: The spread of a cancer from the primary site to somewhere else via the bloodstream or the lymphatic system.

Microcalcifications: Tiny deposits of calcium in the breast, which can show up on a mammogram. Certain patterns of microcalcifications are sometimes a sign of breast cancer.

Mitosis: The process of cell division.

Mutation: A change in the number, arrangement, or molecular sequence of a gene.

Necrosis: The death of an individual cell or groups of cells in living tissue. Sometimes seen in carcinomas.

Needle biopsy: Use of a needle to extract cells or bits of tissue for microscopic examination.

Nodal status: The presence or absence of cancer in the lymph nodes of the armpit. A woman with cancer in one or more nodes is node positive, or node +ve. A woman with no cancer in her nodes is node negative, or node -ve.

Nonpalpable cancer: Cancer in breast tissue that can be seen on mammograms but that cannot be felt.

Oncogene: A gene which, functioning abnormally, encourages normal cells to turn cancerous.

Oncologist: A doctor who specialises in treating cancer.

Oncology: The study of the biology and physical and chemical features of cancers. Also the study of the cause and treatment of cancers.

Oncology nurse: A registered nurse who is educated in the care of people with cancer.

Overall survival: The time from the primary treatment of the breast cancer to death.

p53: A protein which, when the gene for it is damaged, leads to an increased risk of breast cancer.

Palliation: The alleviation of symptoms due to the underlying cancer, without prospect of cure.

Palpation: Use of the fingers to press body surfaces, so as to feel tissues and organs underneath. Palpating the breast for lumps is a crucial part of a physical breast examination.

Primary breast tumor: Tumor arising in the breast.

Prognosis: An estimate of what is likely to happen in the future.

Prognostic factors: Factors that are associated with a better or worse outcome of the disease. They are not the same as causes.

Progression: The continuing growth of the cancer.

Prophylactic mastectomy: Surgery to remove a breast that is not known to contain breast cancer, for the purpose of reducing an individual's cancer risk.

Rad: An old unit of radiation, which stands for radiation absorbed dose, and is superseded by the Gray. 1 Gray = 100 rads.

Radiation: Energy carried by waves or by streams of particles. Various forms of radiation can be used in low doses to diagnose disease and in high doses to treat disease. See X-rays.

Radiation oncologist: A doctor who specializes in treating cancer with radiation. Also known as a radiotherapist.

Radiographer: A technician who gives radiotherapy prescribed by a radiation oncologist.

Radiologist: A doctor with special training in the use of x-rays (and related technologies such as ultrasound) to image body tissues and to treat disease.

Radiotherapy: The use of radiation, usually x-rays or gamma rays, to kill tumour cells.

Remission: A reduction or disappearance of the symptoms of cancer. It can be partial or complete.

Risk: A measure of the likelihood of some uncertain or random event with negative consequences for human life or health.

Risk factors (for cancer): Conditions or agents that increase a person's chances of getting cancer. Risk factors do not necessarily cause cancer; rather, they are indicators, statistically associated with an increase in likelihood.

Screening mammogram: Breast x-ray used to look for signs of disease such as cancer in people who are symptom-free.

Secondary tumor: A deposit of breast cancer away from the breast (such as in the lung, bone or lymph node). See metastasis.

Segmentectomy: The excision of an entire segment of the breast.

Sonogram: The image produced by ultrasound.

Sonographer: A technician trained in performing ultrasounds.

Specimen X-ray: An X-ray of a surgically removed specimen to confirm that a mammographically detected cancer has been removed.

Staging: Refers to the allocation of categories (0, I, II, III, IV) to groupings of tumours defined by internationally agreed criteria. Staging helps determine treatment and prognosis.

Stereotactic localization biopsy: A technique that employs three-dimensional x-ray to pinpoint a specific target area. It is used in conjunction with needle biopsy of nonpalpable breast abnormalities.

Tumor: An abnormal growth of tissue. It may be localized (benign) or invade nearby tissues (malignant) or distant tissues (metastatic).

Tumor suppressor gene: A gene that usually prevents cancers growing. When it is not functioning normally, tumors can grow. Examples include p53 in breast cancer, RB protein in retinoblastoma and possibly BRCA1 in breast cancer. Also known as an anti-oncogene.

Tumor type: The overall cell pattern of the tumor.

Ultrasound: The use of sound waves to form a picture of internal tissues.

Vascular infiltration: Invasion by cancer cells of lymphatics or veins. It is a sign that the tumor is likely to spread.

X-ray: A high-energy form of radiation. X-rays form an image of body structures by traveling through the body and striking a sheet of film. Breast x-rays are called mammograms

Bibliography

- Barton, Mary B., Russell Harris, and Suzanne W. Fletcher. "Does This Patient Have Breast Cancer?" *Journal of the American Medical Association (JAMA)*, 212 (13): 1270-1280 (06 Oct 1999).
- Bassham, Christopher B. Visualizing Early-Stage Breast Cancer Tumors in a Mammographic Environment Through a 3-Dimensional Mathematical Model, MS Thesis, AFIT/GOA/ENS/99M-01, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.
- . Computer Visualization of Breast Cancer Tumor Models. Specialty Study, Air Force Institute of Technology (AU), Department of Operational Sciences, Wright-Patterson AFB OH, June 1999.
- "Breast Cancer—Early Detection is Helping Save Lives," Chain Drug Review, 21 (6): 5 (15 March 1999).
- Coffey, Donald S. "Self-organization, Complexity and Chaos: The New Biology for Medicine." Article, n. pag. <http://medicine.snu.ac.kr/CGMS/self.html>. 10 April 1999.
- "Computers May Help Radiologists Detect Breast Cancer," Women's Health Weekly (14 June 1999).
- Davis, P.L. and K.S. McCarty, Jr. "Sensitivity of Enhanced MRI for the Detection of Breast Cancer: New, Multicentric, Residual, and Recurrent," European Radiology, 1997; 7 (Supp. 5): S289-S298.
- "Digital Mammography Has Promising Future," Women's Health Weekly (14 June 1999).
- "FDA Approves New Breast Cancer Imaging Device," Food and Drug Administration Talk Paper: T99-18 (19 April 1999).
- Hayes, Daniel F., ed. Atlas of Breast Cancer. Hong Kong: Mosby Europe Ltd., 1993.
- "Insight Awards to Stamp Out Breast Cancer," National Cancer Institute: PAR-99-128 (13 July 1999).
- Iynegar, S. Sitharama. Computer Modeling of Complex Biological Systems, Boca Raton, FL: CRC Press, 1984.

- "MRI Detects Breast Tumors Missed by Mammography," Reuters Health excerpt from Radiology, 1999; 212: 543-549.
http://www.oncolink.upenn.edu/cancer_news/reuters/1999/jul/cl07299n.html (29 July 1999).
- "MRI Tops Mammogram, Ultrasound in Detecting Rare Cancer," Medical Industry Today. Medical Data International, Inc. (11 May 1999).
- "MRI vs. Mammography and Ultrasound for Cancer," Women's Health Weekly (10 May 1999).
- Muller, Pierre-Alain. Instant UML. Birmingham, UK: Wrox Press Ltd., 1997.
- National Cancer Institute (U.S.). Office of Cancer Communications. The Breast Cancer Digest: A Guide to Medical Care, Emotional Support, Educational Programs, and Resources (DHHS/NIH 84-1691). 2nd ed. Bethesda, MD: U.S. National Cancer Institute, 1984.
- . The National Strategic Plan for the Early Detection and Control of Breast and Cervical Cancers. Atlanta, GA: U.S. Dept. of Health and Human Services, Public Health Services, Centers for Disease Control and Prevention, 1993.
- "New York Hospital Offers Advanced Cancer Detection," Cancer Weekly Plus, (08 February 1999).
- Palmari, Jacqueline, *et al.* "Topographical Analysis of Spatial Patterns Generated by a Cellular Automaton Model of the Proliferation of a Cancer Cell Line *in Vitro*," Analytical Cellular Pathology, 1997; 14: 75-86.
- Pitot, Henery C. Fundamentals of Oncology. New York: Marcel Dekker, Inc., 1978.
- Pressman, Roger S. Software Engineering: A Practitioner's Approach, 4e. New York: McGraw-Hill, 1997.
- Rosner, B. and G.A. Colditz. "Nurses' Health Study: Log-incidence Mathematical Model of Breast Cancer Incidence," Journal of the National Cancer Institute, 1996; 88: 359-364.
- Saarela, A.O., *et al.* "Mammographic and Ultrasonographic Findings in Bilateral Breast Cancer: A Comparative Study," European Radiology, 1998; 8: 634-638.
- Saltus, Richard. "Cancer: Turning the Corner: New Drugs, New Hope," The Boston Globe, 03 August 1999.
- Schroeder, Will, Ken Martin, and Bill Lorensen. The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. Upper Saddle River, New Jersey: Prentice Hall PTR, 1996.

Schwab, E. D. and K. J. Pienta. "Cancer as a Complex Adaptive System." Excerpt from Medical Hypothesis; 1996; 47: 235-241.
<http://medicine.snu.ac.kr/CGMS/CANCER.html> (14 April 1999).

Simonetti, Giovanni, *et al.* "What's New in Mammography," European Journal of Radiology, 1998; 27: S234-S241.

"Stages of Breast Cancer Development: Normal to Metastatic Disease," National Cancer Institute Public Affairs Release: PA-99-162 (02 September 1999).

Tan, Wai-Yuan. Stochastic Models of Carcinogenesis. New York: Marcel Dekker, Inc., 1991.

Voiculetz, Nicolae, *et al.* Modeling of Cancer Genesis and Prevention. Boston: CRC Press, 1991.

Vita

Captain Bruce C Jenkins was born 29 August 1961 in Bismarck, North Dakota. After graduating from Bismarck high school in June 1979, he enlisted in United States Air Force and attended Basic Military Training School at Lackland Air Force Base (AFB), Texas. Jenkins was subsequently trained as an F-4 Weapon Control Systems (WCS) apprentice at Lowery Air Force Base, Colorado. His first assignment took him to the 35th and 37th Component Repair Squadrons, George AFB, California, where in 1981 he was promoted to Senior Airman below-to-zone. In October 1982, he was assigned to the 37th Tactical Fighter Wing, Spangdahlem Air Base, Germany, where he performed WCS maintenance on the F-4E and F-4G in both flight line and back-shop environments. Temporary duties took him to Skullthorpe, England, and Zaragoza, Spain. During the nearly ten years he served in Germany, he was promoted to the rank of Technical Sergeant. He also served as chief of the 52nd Component Repair Squadron's WCS Element, where he managed a three-shift field shop and radar calibration operation. He completed a Bachelor of Science degree in computer science through the University of Maryland in January 1992. That same year, he retrained into the communications and information career field and was assigned to the Space and Warning Systems Center, Peterson AFB, Colorado. While Superintendent of the Network Software Analysis Branch, he was promoted to Master Sergeant. In August 1995, he was commissioned a Second Lieutenant after attending Officer Training School at Maxwell AFB, Alabama. He received follow-on Basic Communications Officer Training at Keesler AFB, Mississippi. His first assignment as a commissioned officer took him to Headquarters, Air Force Office of Special Investigations (AFOSI), Bolling AFB, Washington, D.C.; and Andrews AFB, Maryland. At AFOSI he was a software configuration manager and then project manager of the Crime and Counterintelligence, Terrorism Information System. Captain Jenkins was accepted to AFIT in 1998. Upon graduation, he will be assigned to the Air Force Wargaming Institute, College of Aerospace Doctrine, Research, and Education, Maxwell AFB, AL, where he will serve as Chief of the Joint Campaign Simulation Branch.

Permanent contact (e-mail):
bcjenkins@acm.org